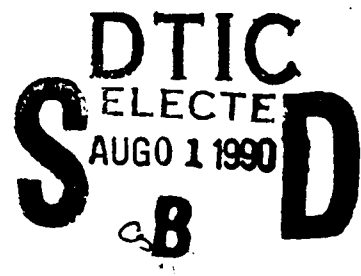


AD-A224 678

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE August 1990		3. REPORT TYPE AND DATES COVERED Thesis/Dissertation	
4. TITLE AND SUBTITLE An Evaluation of a Modified Simulated Annealing Algorithm for Various Formulations				5. FUNDING NUMBERS	
6. AUTHOR(S) James Scott Shedden					
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AFIT Student at: Arizona State University				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/CI/CIA - 90-020D	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFIT/CI Wright-Patterson AFB OH 45433				10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES					
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for Public Release IAW AFR 190-1 Distribution Unlimited ERNEST A. HAYGOOD, 1st Lt, USAF Executive Officer, Civilian Institution Programs				12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words)					
					
14. SUBJECT TERMS				15. NUMBER OF PAGES 208	
				16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED		18. SECURITY CLASSIFICATION OF THIS PAGE		19. SECURITY CLASSIFICATION OF ABSTRACT	
				20. LIMITATION OF ABSTRACT	

AN EVALUATION OF A MODIFIED SIMULATED ANNEALING ALGORITHM
FOR VARIOUS FORMULATIONS

by

James Scott Shedden

A Dissertation Presented in Partial Fulfillment
of the Requirments for the Degree
Doctor of Philosophy

ARIZONA STATE UNIVERSITY

August 1990

TABLE OF CONTENTS

	Page
LIST OF TABLES	vii
LIST OF FIGURES	viii
I. INTRODUCTION	1
II. BACKGROUND	4
A. Combinatorics	5
B. Combinatorial Optimization	7
C. Computational Complexity	8
D. Industrial Engineering Problems	11
E. The Appeal of Simulated Annealing	13
F. Purpose of the Study	14
III. REVIEW OF RELATED LITERATURE	15
A. Computational Complexity	15
B. The Traveling Salesman Problem	21
C. Heuristic Approaches	23
D. Simulated Annealing Background	27
IV. SIMULATED ANNEALING'S SUITABILITY FOR INDUSTRIAL ENGINEERING PROBLEMS	44
A. Features of the Algorithm	46
B. Requirements of a Dynamic Shop Floor Environment	50
C. Ways to Modify the Algorithm	51
V. A MODIFIED SIMULATED ANNEALING FORMULATION	57
A. The General Structure	57
B. Using Simulated Annealing in an Applied Setting	70
C. Parameter Settings	71
D. Cooling Schedules Employed	72
E. Formulation of a Problem	74
VI. APPLICATION OF THE MODIFIED SIMULATED ANNEALING ALGORITHM TO THE JOB SHOP SCHEDULING PROBLEM ..	90
A. Job Shop Formulation	90
B. Job Shop Scheduling Results	94



For	
ed	
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

	Page
VII. APPLICATION OF THE MODIFIED SIMULATED ANNEALING ALGORITHM TO THE FLOW SHOP SCHEDULING PROBLEM.	101
A. Flow Shop Formulation	101
B. Simulated Annealing Versus Palmer's Heuristic	102
C. Simulated Annealing Versus Johnson's Algorithm	109
D. Large Flow-Shop Problems	114
VIII. SIMULATED ANNEALING'S POTENTIAL FOR OTHER INDUSTRIAL ENGINEERING PROBLEMS	121
A. Types of Problems Appropriate for Simulated Annealing	121
B. Example Formulations	122
C. Faster Implementations	134
IX. CONCLUSION	142
A. Summary	142
B. Recommendations for Future Work	145
REFERENCES	147
APPENDIX	
A - TSP Program Code	154
B - Assignment Problem Code	158
C - Zero-One Programming Code	161
D - Minimum Cut Program Code	164
E - Flow-shop Problem Code	167
F - Continuous Function Program Code	170
G - Job-Shop Scheduling Program Code	172
H - Additional Formulations	176

LIST OF TABLES

Table	Page
3.1 Time Requirements	17
3.2 Increase in Instance Size	17
3.3 A Biased Random Walk	42
4.1 Simulated Annealing Analogy	47
6.1 Job Shop Results	94
6.2 Job Shop Results for Alternate Acceptance Functions	98
7.1 Flow Shop Results Using Simulated Annealing	103
7.2 Alpha Sensitivity for Flow Shop Problems	107
7.3 Flow Shop Results for Large Problems	108
7.4 Flow Shop Evaluation for Known Optimal Solutions.	109
7.5 Flow Shop Results for Alternate Acceptance Functions	113
7.6 Flow Shop Results from the CRAY Computer	115
7.7 Flow Shop Results, Large Instances (on a CRAY) ..	118
8.1 Results of the Assignment Problem	129
H.1 Summary of Computational Results	189
H.2 Repeated Application of the 2-Opt Algorithm	191
H.3 Results From Literature	193
H.1 Results for the Traveling Salesman Problem	196
H.2 Results for Large Traveling Salesman Problems ...	197

LIST OF FIGURES

Figure	Page
1.1 Shop Floor Scheduler Formulation	3
3.1 Pseudo PASCAL Program for Simulated Annealing ...	37
3.2 Typical Simulated Annealing Algorithm Performance	39
3.3 Random Walk	40
4.1 A General Structure for the Simulated Annealing Algorithm	49
5.1 A Modified Structure for the Simulated Annealing Algorithm	59
5.2 Modification Impacts	61
5.3 Probability of Acceptance - Standard Form	63
5.4 Probability of Acceptance - Uniform Version.....	64
5.5 FS1 Probability of Acceptance	66
5.6 FS2 Probability of Acceptance	67
5.7 Pseudo Code for the Modified Simulated Annealing Algorithm Applied to the TSP	70
5.8 Swapping Permutation for Zero-One Programming Problem	77
5.9 Reversal Permutation for the Traveling Salesman Problem	78
5.10 Relationship Between Probability of Acceptance and Ratio ($-\delta C/T$)	82
5.11 Relationship Between Probability of Acceptance and Ratio ($-\delta C/T$) Continued	83
6.1 Shop Floor Scheduler	91
6.2 Job Shop Results	97
6.3 Job Shop Results for Alternate Acceptance Functions	100
7.1 Flow Shop Results	106
7.2 Flow Shop Results (XT)	112
7.3 Flow Shop Results, 40x40 Problem	117
7.4 Flow Shop Results (on CRAY)	120
8.1 Pseudo Code for Simulated Annealing Algorithm Applied to the TSP	124
8.2 Pseudo Code for an Assignment Problem	128
8.3 Assignment Problem Results	132
8.4 A Classification of Parallel-Machine Models	137

ABSTRACT

Many combinatorial optimization problems are too large to allow exact solution methods to give results in a reasonable amount of time. An alternative is the use of heuristics that sacrifice a degree of optimality in return for timelier solutions. One such heuristic, simulated annealing, is a form of iterative improvement that probabilistically accepts less optimal configurations. This procedure allows the algorithm to escape local minima (or maxima) in its search for a global minimum.

This paper presents modified simulated annealing formulations of common industrial engineering problems. The modified algorithm behaves like a biased random walk that can be tailored to suit a user's particular bias set. Modifications to the standard application include: an expert system front end, a constraints module, a user interrupt capability, and the creation of alternative acceptance functions. Recent availability of low cost computational speed makes this method a very attractive approach to obtaining "on-time" solutions to many combinatorial optimization problems.

90 07 31 074

I. INTRODUCTION

The availability of low cost computational speed has made it possible to implement heuristic algorithms that provide near optimal "on-time" solutions. Specifically, the new 80386 and 486i based processors and the reduced instruction set architectures provide increased speeds at relatively low cost. These fast machines allow us to take advantage of a new optimization technique that trades elegance for speed. Simulated annealing is a form of local search that results in a biased random walk in its search for an optimal solution. The flexibility of this algorithm makes it possible to attach an expert system, a constraints module, and a user interface to tailor the algorithm to a particular user bias. Each bias set will yield a slightly different answer. The algorithm is also capable of running for a variable length of time. One scenario shows a dedicated processor running the simulated annealing algorithm continuously with users querying the system and extracting solutions as they are needed. The more processing time available the greater the likelihood of obtaining better solutions. The result of this system is "on-time" solutions that are acceptable to the users' particular set of biases.

Figure 1.1 shows the modified simulated annealing algorithm. There are four modification to the standard formulation. First, the user has the option of selecting

from a family of acceptance functions for the particular problem at hand. These functions trade off computational time for solution quality. Second, parameter settings can be input directly or a front end expert system can be queried, resulting the selection of appropriate values. Selection is made based on the problem size, computational power, and processing time available. Third, a constraints module allows the user to input restrictions that eliminate unacceptable solutions. Finally, there is a user interrupt capability that allows the algorithm to react to real time changes in the problem domain.

This paper introduces formulations for the job shop and flow shop problems. These formulations are then used to perform parametric analysis and to evaluate the alternative acceptance functions. Parametric analysis is performed to establish efficient parameter settings for the job shop and flow shop problems. Under certain conditions two of the alternative acceptance functions outperform the standard formulation.

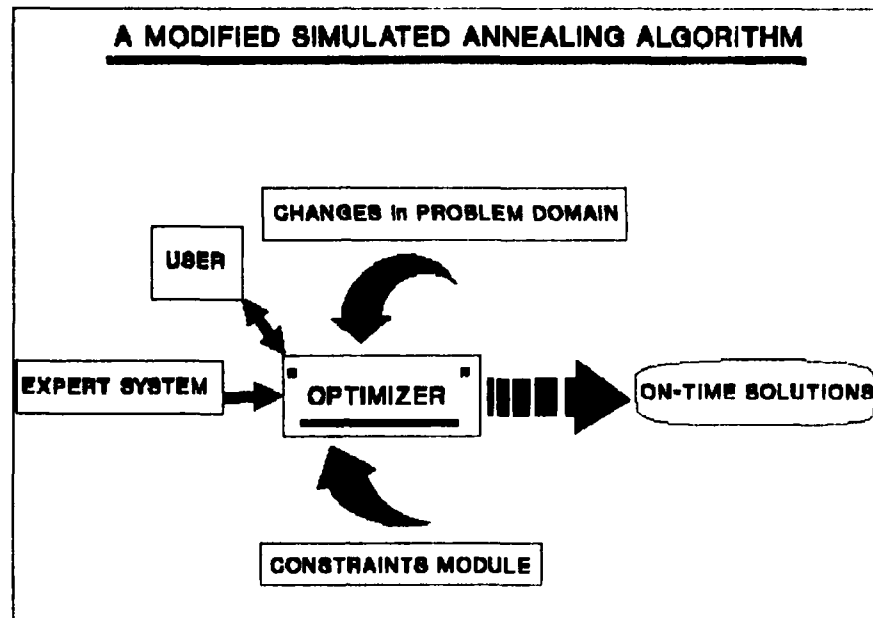


Figure 1.1 Shop floor scheduler formulation.

This paper presents background material in Chapter II, a literature review in Chapter III, and Chapter IV examines the suitability of simulated annealing for common industrial engineering problems. The modified algorithm is presented in Chapter V. A job shop formulation and results for the modified algorithm are shown in Chapter VI. Chapter VII shows the flow shop formulation and its results. Finally, potential for other industrial engineering problems, as well as results are shown in Chapter VIII.

II. BACKGROUND

Simulated annealing is a general purpose approximation algorithm applicable to many combinatorial optimization problems. It behaves like a local search that probabilistically accepts transitions away from optimality, thus allowing it to escape local optima in its search for the global optimum. Therefore, unlike local search, one of the algorithm's key features is its independence of initial configurations. Other strengths are the algorithm's flexibility, enabling it to be formatted for many diverse types of problems, and its ability to solve large problems. The algorithm has a short history of seven years with very little concentration on industrial engineering problems. Since many of the scheduling problems are types of combinatorial optimization problems, there is great opportunity to exploit the power of this heuristic for industrial engineering applications.

A. Combinatorics

One of the fastest growing fields of modern mathematics is combinatorics. This discipline is concerned with the study of arrangements, patterns, designs, assignments, schedules, connections, and configurations. A broad definition is offered by Riordan (1958) as anything enumerative. A major reason for the rapid growth is its wealth of applications; to computer science, communications, transportation, genetics, experimental design, scheduling,

and more (Roberts, 1984). People in numerous industries are faced with combinatorial problems. A shop supervisor must find an optimal allocation of workers to machines. An industrial engineer must choose, among many production schedules, one that will best support management's goals. Much of the growth of combinatorics has been accompanied by commensurate developments in computers. Today's fast computers make it possible to implement algorithms for practical combinatorial problems that were not feasible to implement on slower computers. The result has been an increased emphasis on developing fast algorithms to find solutions to these problems. Thus, it is difficult to separate combinatorial mathematics from computing.

1. History

Although most of the impetus of combinatorics has been in modern times, it is an old branch of mathematics. Permutations, ordered arrangements, were known in China before 1100 B.C. The binomial expansion [the expansion of $(a + b)^n$] was known to Euclid about 300 B.C. for the case where $n=2$. In 1100 A.D., Rabbi Ibn Ezra knew the formula for the number of combinations of n things taken r at a time. Shortly thereafter, Hindu, Arab, and Chinese works mentioned binomial coefficients in primitive ways (Roberts, 1984).

In more recent times, the seventeenth century scholars Pascal and Fermat studied combinatorial problems associated

with gambling - figuring out odds. This work laid the foundation for probability theory in the eighteenth century by Laplace. He gets credited with defining probability in terms of the number of favorable cases. Also, Euler invented graph theory and Bernoulli published the first book presenting combinatorial methods. Leibniz in his "ars combinatoria" is credited with originating the subject titled combinatorial analysis (Riordan, 1958). In the nineteenth century Hamilton used combinatorial techniques to study puzzles and games. In modern times the techniques have had far reaching effects on the fields of computer science, transportation, information processing, industrial planning, electrical engineering, experimental design, coding, genetics, and a variety of other important areas (Roberts, 1984).

2. Problems

There are three basic types of problems: existence problems, counting problems, and optimization problems (Roberts, 1984).

Existence Problems. The existence problem attempts to answer the question: Is there at least one arrangement of a particular kind (Roberts, 1984).

Counting Problems. A counting problem asks: How many arrangements are there (Roberts, 1984)?

Optimization Problems. The combinatorial optimization problem (the focus of this paper) is concerned with finding

the 'best' or optimal solution among a finite or countably infinite number of alternative solutions (Papadimitriou & Steiglitz, 1982).

B. Combinatorial Optimization

"The process of optimization lies at the root of engineering, since the classical function of engineers is to design new, better, more efficient, and less expensive systems as well as to devise plans and procedures for the improved operation of existing systems (Reklaitis, Ravindran, & Ragsdell, 1983)." A gas pipeline flow problem is used to illustrate an optimization problem. The problem of designing a pipeline system involves many variables including the size of the various links between junctions that will minimize the total cost of construction and operation. A modest network of 40 links and 7 possible pipe diameters has 7^{40} potential networks. This is a very large number, 6.367×10^{33} , when attempting to find an optimum (best, maximum, minimum).

As already pointed out, progress in solving combinatorial optimization problems has gone hand in hand with the developments with the computer. Integer, linear and non-linear programming, and dynamic programming have experienced major breakthroughs in recent years (Aarts & Korst, 1989). Even with the amazing speeds attained by today's computers, there are limitations as to what can be accomplished (Roberts, 1984). One approach to solving this

problem is to evaluate each possible network configuration. This is the exhaustive enumeration approach. Returning to our problem, if a computer is capable of analyzing 1 billion different networks per second (one each nanosecond), it will still take 1.9×10^{17} years to solve the problem. This problem is not tractable using current high-speed computers (Roberts, 1984). Among all of the combinatorial optimization problems the traveling salesman problem is probably the best known (Lawler et al., 1985). This is discussed more thoroughly in the next section. Much of modern combinatorial optimization is concerned with creating algorithms that solve efficiently these types of problems. There are two broad approaches to solving these problems: one can attempt to find optimality at the risk of great computational time, or one finds a quickly attainable solution at the risk of sub-optimality (Aarts, & Korst 1989). For practical reasons, it is very important to analyze these algorithms in terms of their speed and memory requirements. Before implementation on a machine, we want an assurance that the computations can be carried out in a reasonable amount of time and within the available storage capacity of the machine.

C. Computational Complexity

As pointed out, before running a program we must know if it will run in a reasonable amount of time and require no more than a reasonable amount of memory. To measure the

expense of a program, we try to calculate a cost function or complexity function f . For our purposes, f will measure the cost in terms of time required to find the solution, as a function of the input size n . As an example we might ask how many operations are required to multiply two square $n \times n$ matrices. The number of operations is $f(n)$. Since there can be a significant difference in program running times depending on the efficiency of the code and the speed of the computer, the focus will be on the algorithm and the number of calculations required.

There are classes of problems depending on performance of algorithms designed to solve them. These are discussed more thoroughly in Chapter III. The class "P" includes problems for which algorithms with polynomial time behavior have been found. The class "NP" is essentially the set of problems for which algorithms with exponential time behavior has been found (French, 1982). NP-hard and NP-complete are also terms that are used to define the "NP" class of problems for which it is unlikely there will be good algorithms.

One problem that is of particular interest due to its simplicity of definition, but great difficulty in solving is the Traveling Salesman Problem (TSP). This is one of the most studied problems in combinatorial optimization. The TSP can be stated as follows. A salesman, starting from a city must travel to each of $(n-1)$ other cities once and only

once and then return to the starting location. The problem is to determine the order that will result in the shortest path route. Distance can be replaced by other performance measures such as time or cost (Phillips & Garcia-Diaz 1981). An analogous industrial engineering problem would be the routing of a robot, in an automated warehouse, that must visit n different locations to fill an order. To estimate the computational complexity we must consider the enumeration of all possible routes and the calculation of the associated cost of each route. There are n ways of picking the first city on the route, $n-1$ ways of picking the second, and so on. This results in $n!$ possible routes, when we must pick the starting location. For a problem instance that has 25 cities this becomes $25!$ or 1.55×10^{25} , a very large number. Although work has been reported on a problem having 6,406 cities (Kirkpatrick, 1984), the largest problem known to have been solved (proven optimal) has 318 cities (Lawler et al., 1985). This was solved by Crowder and Padberg (1980). The huge amount of computational time required to find the precise optimal solution to many of these NP-hard problems has led researcher in recent years to focus on heuristic approaches to finding near-optimal solutions (Tucker, 1984). A heuristic is simply a rule-of-thumb method or approach to solving the problem. The best solution is not guaranteed, but is sacrificed as a tradeoff

for saving processing time. Examples of heuristics include local search and randomization algorithms.

D. Industrial Engineering Problems

Many of the problems encountered in industrial engineering concern the optimal scheduling or sequencing of jobs. In fact, the basic terminology of scheduling theory arose in the processing and manufacturing industry (French, 1982). The basic problem is to schedule n jobs on m machines. Depending on the scenario and the performance objective there are numerous variations to the general job-shop scheduling problem. Virtually any manufacturing firm that is not mass producing a single product will encounter some form of job-shop scheduling problem. Each product will have its own route through a set of machines. An automated guided vehicle (agv) must sequence a route to its various pick-up and delivery points in the shortest time.

The goal of scheduling varies from one industry to another and often from one day to another. The objective might be to maintain an equal level of activity throughout all departments. Another goal might be to make a suspense date or to simply finish all work as soon as possible (French, 1982).

As pointed out earlier, problems that are hard to solve in a timely manner are classified as "NP-hard". Most of the scheduling problems fall into this classification. In fact, informed mathematical opinion states that there will never

be any easy solutions for these type problems (French, 1982).

Schedules are often very dynamic. One schedule might be optimal at one point in time and the arrival of a "hot" job, necessitating a revision to the schedule, results in a sub-optimal plan. Other dynamic factors include the rerouting of parts, machine breakdowns, rework, worker accidents, and more. Although these problems are very hard to solve to optimality, they nevertheless do arise in industry and their solutions are sought. Therefore, there is a need for some form of real-time scheduling approach to address the dynamic nature of scheduling problems.

E. The Appeal Of Simulated Annealing

One method, termed simulated annealing, offers a heuristic approach that is applicable to a very broad range of combinatorial optimization problems (including scheduling). By setting the various parameters to different values, this algorithm is tailored to the particular problem at hand. The recent emphasis in parallel processing provides opportunities for exploiting some of the features of simulated annealing. Finally, in implementing this approach there is potential for establishing a dedicated processor constantly searching for "better" solutions. Schedulers can query the processor whenever they are in need of a current "best" solution.

F. Purpose Of The Study

The purpose of this effort is to present a structured approach to solving various combinatorial optimization problems. The methodology uses a Monte Carlo technique, common to the field of quantum mechanics, titled simulated annealing. In practical implementations this is viewed as a general purpose approximation technique or heuristic. Several algorithms are formulated that are designed to solve a variety of combinatorial optimization problems. The performance of the various algorithms are analyzed relative to processing time used and percentage of optimality achieved.

III. REVIEW OF RELATED LITERATURE

A. Computational Complexity

To introduce this section a hypothetical situation will be presented. Suppose you are asked to develop an algorithm that will solve a scheduling problem. The objective is to sequence jobs on a machine(s) in a way that minimizes total processing time. The order of the jobs will determine the amount of setup time to be included in each job's total processing time. Depending on a further description of the problem, one of three broad classes of algorithms may be attempted.

Constructive algorithms are used to find quickly the optimal solution. Examples are Johnson's Algorithm (French, 1982) for the two machine flow-shop problem, or the greedy algorithm (Forgionne, 1986) for a minimal spanning tree problem. Constructive techniques, however, have been found for only the simplest of problems (French, 1982). The next best alternative is to use implicit enumeration. Here, the search space is pruned using some approach tailored to the problem being considered. The most common methods are forms of dynamic programming and branch-and-bound (Hillier & Lieberman, 1980). If we are unable to develop a suitable recursion for dynamic programming or lower bound for branch and bound, these techniques can not be used. As a result we are forced to use explicit enumeration, or simply evaluate all possible configurations in the solution space.

Problems that are appropriate for implicit enumeration can involve great computational effort. French (1982) estimates that at a rate of 1 mathematical operation per micro-second the dynamic programming solution to scheduling 40 jobs on one machine will require more than four years to solve. Ignall and Schrage (1965) report that their algorithm for scheduling jobs on three machines requires approximately twice the time for $(n+1)$ jobs as for n jobs. Thus if it takes one second to solve an n job problem it will take 2^n seconds to solve the $(n+r)$ job problem. With $r=20$, this translates into 12 days (French, 1982). The results for problems requiring explicit enumeration are exponentially worse. For these reasons we are encouraged to employ constructive methods whenever possible.

An algorithm's time complexity function $f(v)$ yields the maximum number of operations required to solve an instance of size v . In practice, the time complexity function is not solved completely, rather an indication of its growth is made as the problem size increases. To see the dramatic growth in computational requirements for various time complexity functions examine Tables 3.1 and 3.2 taken from French (1982).

Table 3.1

The Time Requirements of Algorithms with Certain Time Complexity Functions Under the Assumption that One Mathematical Operation Takes One Micro-second. (Garey, 1979)

Time Complexity Function	v				
	10	20	30	40	60
v	.00001 s	.00002 s	.00003 s	.00004 s	.00006 s
v ²	.0001 s	.0004 s	.0009 s	.0016 s	.0036 s
v ⁵	.1 s	3.2 s	24.3 s	1.7 m	13 m
v ¹⁰	2.7 h	118.5 d	18.7 y	3.3 c	192 c
<hr/>					
2 ^v	.001 s	1.0 s	17.9 m	12.7 d	366 c
3 ^v	.59 s	58 m	6.5 y	3855 c	1.3x10 ¹³ c
v!	3.6 s	770 c	8.4x10 ¹⁶ c	2.5x10 ³² c	2.6x19 ⁶⁶ c

s=seconds, m=minutes, h=hours, d=days, y=years, c=centuries

Table 3.2

Increase in Instance Size Solvable in a Given Time for a Thousand-fold Increase in Computing Speed. (Garey, 1979)

Time Complexity Function	Size of instance solved in given time on slow computer	Size of instance solved in same time on computer 1000-times faster
v	v ₁	1000 v ₁
v ²	v ₂	31.62 v ₂
v ⁵	v ₃	3.98 v ₃
v ¹⁰	v ₄	1.99 v ₄
<hr/>		
2 ^v	v ₅	v ₅ + 10
3 ^v	v ₆	v ₆ + 6
		v ₇ + 3 , v ₇ ≤ 10
		v ₇ + 2 , 10 < v ₇ ≤ 30
v!	v ₇	v ₇ + 1 , 30 < v ₇ ≤ 1000

Entries above the dotted lines indicate time requirements for algorithms exhibiting polynomial time behavior, while entries below are for exponential behavior. The increase in time requirements grows much more dramatically for the exponentially bounded algorithms than for the polynomially bounded ones. Table 3.2 shows that algorithms with polynomial time complexity allow multiplicative increases in the instance size for a given gain in computational speed, while those with exponential time complexity allow only an additive increase (French, 1982).

Problems are said to be well solved when we can create an algorithm with polynomial time complexity. Exponential time algorithms are predominantly forms of exhaustive search (French, 1982). Furthermore, polynomial-time algorithms are in better position to take advantage of the increases in computer speeds as technology matures. Lawler et al. (1985) presents a good illustration. Suppose we have two algorithms, one running in time $O(v^3)$, the other in $O(2^v)$, both of which can solve a problem with $v=100$ in an hour. If we get a new computer that is twice as fast, the polynomial-time algorithm can now in one hour solve instances of $v=126$ (a multiplicative factor of 1.26). The exponential algorithm can only obtain an additive increase in instance size, $v=101$.

Up to this point we have restricted our discussion to a classification of the algorithm's performance. A similar classification applies to the type of problem.

1. The Class "P"

We classify a problem as being either 'hard' or 'easy' depending on whether or not it is solvable by an algorithm with polynomial time complexity (Lawler et al., 1985).

"Such a problem can be regarded as tractable if and only if there is an algorithm for its solution whose running time is bounded by a polynomial in the size of its input (Karp, 1972)." Problems for which algorithms with polynomial time behavior have been found belong to the class P. Problems for which algorithms with exponential time behavior have been found belong to the class NP. Since we can always make a polynomial algorithm inefficient to the point where it runs in exponential time we say that P is contained in NP.

For a problem π_1 , to be **polynomially reducible** to π_2 , means that we can reduce π_1 to π_2 in polynomial time. In other words, we can take an instance of π_1 with size v and convert it to an instance of π_2 in some polynomial number of operation $p(v)$. The two instances are also equivalent, in that a yes answer to the first is obtained if and only if a yes answer to the second is obtained (Lawler et al., 1985). The discussion up to this point has laid the foundation for the important concept of NP-completeness.

2. "NP-complete"

"We say that a problem π lying in NP is NP-complete if every other problem in NP is polynomially reducible to π (French, 1982)." This subclass of NP contains the hardest problems in NP. If we can find a polynomial time algorithm for any NP-complete problem, then we have found a polynomial time algorithm for all problems in NP. This concept was originally introduced by Cook (1971) and elaborated on by Karp (1972). Garey and Johnson (1979) list about 300 NP-complete problem types. The real value of this theory is that it provides many straightforward techniques for proving that a given problem is "just as hard" as a large number of other problems that are widely recognized as being hard and have confounded experts for years (Garey & Johnson, 1979). Prior to Cook's theory people had been trying for many years to solve problems that have subsequently been proven to be NP-complete (French, 1982).

In Cook's (1972) work he lists a number of NP-complete problems. These include: 0-1 integer programming, the knapsack problem, job sequencing, max cut, and many more. Later, Garey and Johnson (1979) categorize 300 problems into one of 12 groupings. These include, but certainly are not limited to: traveling salesman, graph partitioning (max flow/min cut), sequencing problems, and flow-shop scheduling problems. These problems will get detailed treatment in later chapters.

Now that we respect the difficulty involved in solving these problems we can move on to looking at one of the NP-complete problems, the traveling salesman problem.

B. The Traveling Salesman Problem (TSP)

The importance of this problem is not for its applications, but for the developments it has inspired. The first appearance of the problem is in a German book by Voight in 1831, but the first use of the term in mathematical circles was in 1931. Flood (1956) is responsible for publicizing it in the math and operations research communities (Lawler et al., 1985). In 1948 John Williams urged Flood to popularize the term TSP at the RAND Corporation. It quickly became the topic of discussion at RAND which was becoming the intellectual center for much of operations research. One reason for the popularity of the problem was its connection with prominent topics in combinatorial problems arising in the new subject of linear programming, namely the assignment and transportation problems. The appearance of 'Solution of a Large-scale Traveling Salesman Problem' (Dantzig, Fulkerson, & Johnson, 1954) was one of the principle events in the history of combinatorial optimization. In this paper they introduced two important new concepts, those of cutting planes and the branch and bound.

Another enumerative method, the recursive technique of dynamic programming, was also applied to the TSP by Bellman

(1962). Due to its enormous storage requirements, this technique can only solve relatively small problem instances (Lawler et al., 1985).

Although, Kirkpatrick (1984) has reported progress on a problem having 6,406 cities, the largest TSP known to be solved (Crowder & Padberg, 1980) to optimality contains 318 cities.

Similar to the TSP, the assignment problem uses the same cost matrix as the TSP. Here the objective is to choose n elements from the $n \times n$ matrix, one from each column and row, in a way that minimizes the sum of the elements chosen. Since there are $n!$ possible ways of making the choices, something other than an enumerative procedure must be employed. Birkhoff (1946) showed how to formulate this as a linear programming problem and solve it efficiently.

By the 1960s the people appreciated the distinction between hard problems such as the TSP (requiring enumerative algorithms) and easy ones like the assignment problem, for which polynomial time algorithms exist (Lawler et al., 1985).

The alternative to optimization is the use of a heuristic to provide a near optimal solution at much less computational cost.

C. Heuristic Approaches

Heuristics are, very simply, rule-of-thumb methods for judging the relative merits of alternative actions (Pearl, 1984).

1. Purpose of Heuristics

The purpose of a heuristic is to find a near optimal solution in the allotted amount of time. They are employed in an attempt to trade-off a degree of optimality for an improvement in speed. Heuristics are only applicable to problems where constructive algorithms do not exist and where implicit enumeration is computationally infeasible. For these cases heuristics are often the only practical approach. Many heuristic methods have been developed with computational requirements proportional to small powers of N , where N represents the instance size (number of cities for a TSP) (Kirkpatrick, Gelatt, & Vecchi, 1983).

Although the primary performance measure of a heuristic is often the quality of the solution, there are other important criteria for judging one's worth. Ball and Magazine (1981) list the following:

- (1) Running Time,
- (2) Ease of Implementation,
- (3) Flexibility, and
- (4) Simplicity.

Running time is often the second most important consideration. Ease of implementation can be very important depending on how much easier one approach is compared to its alternative. Flexibility refers to the algorithm's ability to solve problem variations. A TSP heuristic that can solve only undirected TSPs is clearly inferior to one that can handle both undirected and directed TSPs. Finally, simple algorithms are more appealing to the user than cumbersome ones and more readily lend themselves to various types of analysis (Lawler et al., 1985).

2. Types of Heuristics

Heuristics can be categorized in different ways. One scheme differentiates between applications. Those that can be used for a wide range of problems are general purpose, while those that solve a narrow class of problems are tailored heuristics. Another method of classifying them is to look at their problem-solving methodology. Kirkpatrick, Gelatt, and Vecchi (1983) state that there are basically two strategies for heuristics: **divide-and-conquer** and **iterative improvement**. In divide-and-conquer, the problem is divided into subproblems of a manageable size, then the subproblems are solved. This requires that the subproblems be naturally disjoint. Solutions to the subproblems must then be patched back together. In iterative improvement, the system starts in a known configuration. Some form of rearrangement is applied to the current configuration. If the new

configuration represents an improvement, then it becomes the current configuration. The process continues until no improvements can be found. Since this process often gets stuck in a local (but not global) optimum, it is usually carried out many times with different starting configurations.

One form of divide-and-conquer heuristic we have previously discussed is the branch-and-bound method. By making some heuristic guesses at the most promising branches to explore we can prune the search tree and reduce the time needed to get a solution. The efficiency of the search strategy is directly related to the goodness, or strength, of the heuristic. A weak heuristic will not guide the search well and tends to build a wandering tree, while strong heuristics build narrow and focused search trees (Nilsson, 1980). Suppose, for a scheduling problem, that instead of searching the entire tree we stop as soon as we reach a feasible configuration (a complete schedule). Ashour (1970a, 1970b) has studied this approach. Empirical results for 100 scheduling problems tested show an average efficiency of 95 percent and in no case was the efficiency below 75 percent (French, 1982). Here, efficiency is assumed to refer to optimality.

Local search is a form of iterative improvement. These type of algorithms are discussed extensively by Papadimitriou and Steiglitz (1982). A brief description

will be presented here. Given the current configuration, trial configurations are generated which are closely related (neighbors) to it. Each is examined to see if a better configuration exists. The process stops with a solution that is at least a local optimum. If there is still time remaining until a solution is needed, we can start over with an initial configuration that is more optimal than the previous initial configuration. The process eventually stops and the best solution is used. The definition of 'closely related' varies with different heuristics. Lin and Kernighan (1973) define the 'neighbors of a tour (TSP problems) to be those tours which can be obtained from it by performing a limited number of interchanges of tour edges.

Two comprehensive studies (Golden et al., 1980) and (Adrabinski & Syslo, 1983) evaluate the performance of many heuristic algorithms designed to solve the TSP. Results show the heuristics achieve five to seven percent of optimality in a relatively efficient manner, but further improvements require repeated application.

The next section introduces a generalization of iterative improvement where controlled 'less-optimal' moves are allowed.

D. Simulated Annealing Background

1. Quantum Mechanics Roots

The underlying distribution used in the standard form of simulated annealing has its foundations in the field of quantum mechanics. Fromhold (1981) describes this field as a theory that is used to predict the behavior of atomic and subatomic systems. These systems make up the microscopic domain of nature, while the predictions can be used to understand the macroscopic domain. A quantum-mechanical model features atoms with electrons in certain patterns which constitute the electronic states. These discrete states are characterized by specific values for the energy and angular momentum of the electrons. Collision of the atoms or absorption of electromagnetic radiation promotes the electron in the atom to a higher energy state. Emission of electromagnetic radiation, then follows a transition to a lower energy state. The lowest energy state for the system is termed the ground state (Fromhold, 1981). The concept of a stationary quantum state, first conceived by Niels Bohr (1913), is defined as a condition of a system such that all observable physical properties are independent of time. Each stationary quantum state has a definite energy, but several may have identical or nearly identical levels (Kittel, 1969).

Similarly, statistical mechanics deals with calculating thermodynamic properties of the macroscopic physical system

in terms of the microscopic constituents, such as atoms or molecules (Farquhar, 1964). Described by Schrodinger (1946), statistical mechanics is the central discipline of condensed matter physics, a collection of methods for analyzing aggregate properties of the large number of atoms found in samples of liquids or solids. A thermodynamical description of a system is characterized by relatively few parameters needed to specify completely the thermodynamic state of the system. If such a macroscopic description is possible, in terms of the vastly more numerous parameters describing all the particles of the system, then grouping and averaging these dynamic parameters is necessary (Farquhar, 1964). The term ensemble refers to the grouping of configurations into sets. "In order to calculate thermodynamical properties of a single system the corresponding properties at one instant of each of an ensemble of systems are averaged over this ensemble; these ensemble averages are then assumed to represent the properties of the single system (Farquhar, 1964)."

The probability that the system will be in a particular energy level, $E(\{r_i\})$, at temperature T is given by the Boltzman occupation probability:

$$P(E\{r_i\}) \text{ is distributed } \exp(-E(\{r_i\})/k_B T)$$

where, k_B is Boltzman's constant.

Kirkpatrick, Gelatt, and Vecchi (1983) use this probability as a weighting scheme to enable the calculation of ensemble averages. To illustrate, the number of atoms in one cubic centimeter of matter is on the order of 10^{23} . With such a large number, only the most probable behavior of the system in thermal equilibrium at a given temperature is observed in experiments (Kirkpatrick, Gelatt, & Vecchi, 1983). This is characterized by the average and small fluctuations about the average, when the average is taken over the ensemble of identical systems. In this ensemble, each configuration, defined by the set of atomic positions $\{r_i\}$, of the system is weighted by its Boltzman probability factor, $\exp(-E(\{r_i\})/k_B T)$ (Kirkpatrick, Gelatt, & Vecchi, 1983). Normalizing, we get the Boltzman distribution, which characterizes the probability of being in a state with energy E .

$$P\{E=E\} = 1/Z(T) * \exp(-E/k_B T)$$

Where, $Z(T)$ is the normalization factor (Van Laarhoven & Aarts, 1987).

Although statistical mechanics has historically dealt with thermodynamics it can be applied in more general terms, answering the question: 'Given partial information about a physical system, how are we to make the best prediction of

the results of further measurements of the system?' (Wyllie, 1970).

2. Simulated Annealing's Methodology

In the previous section the Boltzman distribution for characterizing thermal equilibrium was introduced (Wyllie, 1970).

$$P_r \{X=i\} = 1/Z(T) \exp(-E_i/k_b T),$$

where

$$Z(T) = \sum_j \exp(-E_j/k_b T)$$

This gives the probability of a solid being in state i with energy E_i at temperature T , where X is a stochastic variable. $Z(T)$ is the partition function, with the summation extending over all possible states (Aarts, 1989). In this section we will show the distribution's relation to combinatorial optimization problems and why it works.

Metropolis et al. (1953) introduced an algorithm that simulates a collection of atoms at equilibrium for a given temperature. For a combinatorial optimization problem we would have a very large, but finite number of configurations. Let v_r be the number of configurations in state r . Here r refers to a random variable defining the state (energy) of the system. For a TSP we could have 20

different routes each with a distance of 50 miles. We must prove that after many moves the ensemble approaches the distribution:

$$V_r \text{ is distributed: } \exp(-E_r/kT)$$

Let $P_{r,s}$ be the probability that we transition from state r to state s . It should be clear that, prior to weighting transitions, $P_{r,s} = P_{s,r}$. A particle is equally likely to move anywhere within the volume. Thus, if r and s differ only by the position of the moved particle, the transition probabilities are equal. Now, assume that $E_r > E_s$, E is the measure of energy. The number of transitions from r to s will be $V_r P_{r,s}$ (all moves to a lower energy state are accepted). Weighting moves to higher energy states by the exponential factor, we get the number of transitions from s to r equal to $V_s P_{s,r} \exp(-(E_r - E_s)/kT)$. The net number of transitions from s to r is:

$$\text{Net} = V_s P_{s,r} \exp(-(E_r - E_s)/kT) - V_r P_{r,s}$$

since $P_{s,r} = P_{r,s}$,

$$\text{Net} = P_{r,s} (V_s \exp(-(E_r - E_s)/kT) - V_r)$$

Therefore, for any two states r and s , where

$$(V_r/V_s) > [\exp(-E_r/kT)/\exp(-E_s/kT)],$$

on the average, there are more transitions from state r to state s . In words, if the ratio of the number of configurations in state r to the number in state s is greater than their ratio of probabilities of being in that state, then there are more transitions from r to s . This can be shown with a simple numerical example:

Let: $V_r = 20$, $V_s = 5$, $E_r = 2$, $E_s = 1$, $T = 1$, $P_{s,r} = P_{r,s} = .5$

$$\text{Prob}(r) = \exp(-2/1) = .14$$

$$\text{Prob}(s) = \exp(-1/1) = .37$$

Then: $V_r/V_s > \text{Prob}(r)/\text{Prob}(s)$

$$20/5 > .14/.37$$

$$4 > .38$$

and: Net R to S : $.5(20) = 10$

$$\text{Net } S \text{ to } R: .5(5(.37) - 20) = -9$$

Thus, there will be more transitions to the lower energy state s . Metropolis et al. (1953) states that when the process is ergodic (any state can transition to any other state, albeit in several moves) and, based on the above weighting scheme, more systems move from r to s , then the ensemble must approach the canonical distribution:

V_r is distributed: $\exp(-E_r/kT)$

In relating this to combinatorial optimization, information theory provides us with a valuable measurement. To be an unbiased estimate, the probability distribution must maximize our degree of uncertainty, measured by the statistic entropy, concerning the occurrence of any configuration (Bonomi & Lutton, 1984). The resulting set of probabilities is known as the Boltzman distribution.

Bonomi and Lutton (1984) present a good illustration using an example of a hidden object in one of n identical boxes. Similarly, these could represent n possible configurations of a particular instance. Let P_i be the probability assigned to each.

$$(1) \quad P_i \geq 0, \quad i = 1, 2, \dots, n,$$

$$(2) \quad \sum_{i=1}^n P_i = 1.$$

With perfect knowledge we know exactly where the object is and our uncertainty would be zero. At the other extreme, with no information, then all possibilities are equally likely, $P_i = 1/n$, implying that our uncertainty is maximal. C. Shannon (1948) introduced a measure of degree of uncertainty:

$$(3) \quad S[P_1, \dots, P_n] = -K \sum_{i=1}^n P_i \ln P_i,$$

where K is an arbitrary positive constant used to fix the units of information. This is also the expression for entropy. E.T. Jaynes (1957) suggests a reinterpretation of statistical mechanics stating it is the result of statistical inference representing "the best estimate that could have been made on the basis of the information available." We know from classical statistics that the expected value of a function is:

$$\langle f(x) \rangle = \sum_{i=1}^n P_i f(x_i).$$

The problem is how to statistically describe the set of admissible states x_i (Bonomi, 1984). The distribution P_i maximizes the uncertainty (3) subject to the available information. To maximize (2), subject to (1) and (3), the method of Lagrange multipliers is used:

$$L = -K \sum_{i=1}^n P_i \ln P_i - \theta \left(\sum_{i=1}^n P_i \right) - \mu \left(\sum_{i=1}^n P_i f(x_i) \right)$$

which reduces to:

$$L = \ln P_i - \theta - \mu(f(x_i))$$

Taking the partial derivative with respect to P_i yields:

$$P_i - e^{-\theta - \mu f(x_i)} = 0, \text{ or}$$

$$(4) \quad P_i = e^{-\theta - \mu f(x_i)}, \quad i=1,2,\dots,n,$$

where θ and μ are the Lagrange multipliers. Substituting (4) into (1) and (2), yields:

$$\sum_{i=1}^n e^{-\theta - \mu f(x_i)} = 1$$

$$\sum_{i=1}^n e^{-\theta} \sum_{i=1}^n e^{-\mu f(x_i)} = 1$$

$$e^{-\theta} = Z(u)$$

$$\text{where } Z(u) = \sum_{i=1}^n e^{-\mu f(x_i)}$$

Therefore, this reduces to the Boltzman distribution:

$$P_i = [1 / \sum_{i=1}^n e^{-\mu f(x_i)}] e^{-\mu f(x_i)}$$

With $f(x_i) = (-E_i/T)$, this is identical to the standard form used in the Metropolis procedure. In this procedure an atom is given a small random displacement and the resulting change in energy is computed, δE . If $\delta E \leq 0$, the displaced configuration is accepted and used as the starting point. Where $\delta E > 0$, the acceptance is treated probabilistically:

$P(\delta E) = \exp(-\delta E/T)$. A random number from a uniform (0,1) distribution is drawn and compared to $P(\delta E)$. If $P(\delta E) \geq UN(0,1)$ then the new configuration is accepted, otherwise the current configuration is retained. Figure 3.1 is a pseudo PASCAL program taken from Aarts and Korst (1989) to illustrate this procedure.

```

procedure SIMULATED_ANNEALING
begin
    INITIALIZE (istart, c0, L0);
    k:=0;
    i:=istart;
    repeat
        for l:=1 to Lk do
            begin
                GENERATE (j from Si)
                if f(j) ≤ f(i) then i:=j
                else
                    if exp(f(i)-f(j))/ck > rand(0,1) then
                        i:=j
            end;
        k:=k+1;
        CALCULATE_LENGTH(Lk);
        CALCULATE_CONTROL(ck);
    until stopcriterion
end;

```

Figure 3.1 Pseudo PASCAL program for simulated annealing.

"It is obvious that the Metropolis procedure is independent of any considerations concerning the microscopic laws governing the transition amongst the set of states. This makes it possible to apply the algorithm in a more

general context including combinatorial optimization problems (Bonomi, 1984)."

Let the probability of finding a system in a particular configuration with energy E be given by the Boltzman-Gibbs distribution:

$$P(\text{conf}) = \exp(-E_{\text{conf}}/T) / \left[\sum_{\text{conf}} \exp(-E_{\text{conf}}/T) \right].$$

Then, the mean energy of the system at equilibrium is given by:

$$(5) \quad \bar{E} = \left[\sum_{i=1}^n E_i \exp(-E_i/T) \right] / \left[\sum_{i=1}^n \exp(-E_i/T) \right].$$

This can be thought of as a conditional expectation for the energy level. The numerator represents the joint probability distribution, while the denominator is the marginal distribution for the configurations.

Cerny (1985) states that all that is needed is an implementation that maintains an equilibrium as T is decreased. Further, one reaches the equilibrium when using the Metropolis procedure after a reasonable number of trials as depicted below.

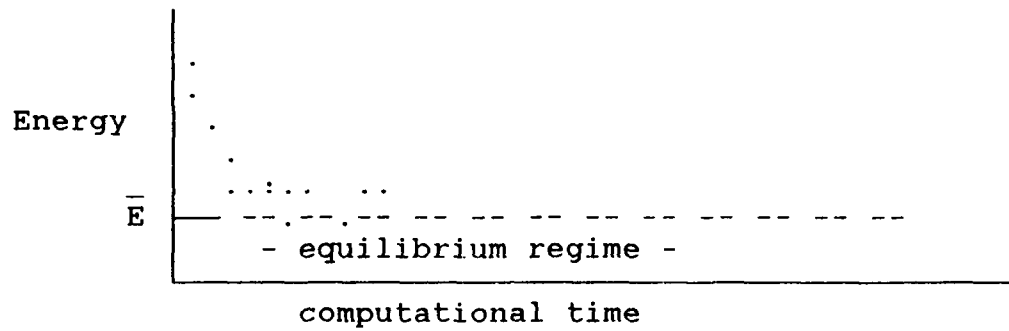


Figure 3.2 Typical simulated annealing algorithm performance.

The result is a biased random walk that favors lower energy states. Therefore, by decreasing T in equation (5) the average energy can be lowered systematically.

3. A Biased Random Walk

Feller (1966) describes a random walk on a number line starting at the origin. A particle moves with probability p one step to the right and with probability q ($q=1-p$) one step to the left. He extends this thought to a gambler that can either win his bet or lose it. Here players A and B each have probability $\frac{1}{2}$ of winning a dollar from the opposing player. If N represents the number of dollars that A has, at any time N can change to $N-1$ or $N+1$, each with probability $\frac{1}{2}$. The process continues until one of the gamblers has won all the money and the opposing gambler is broke. Figure 3.3 depicts this random walk.

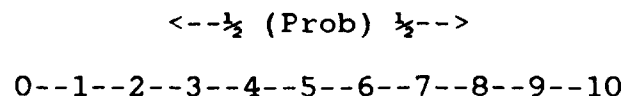


Figure 3.3 Random walk.

Hausner (1971) extends this treatment to look at the initial amount of money each gambler has and their respective probabilities of winning a bet. He lets X_n be the probability that A will win everything when he currently has n dollars. For our game, $X_0=0$ and $X_{10}=1$. For $0 < X < 10$, we get: $X_n = \frac{1}{2}X_{n-1} + \frac{1}{2}X_{n+1}$. This results in a system of nine equations and nine unknowns. Adding the quantity $-\frac{1}{2}X_n$

$- \frac{1}{2}X_{n-1}$ to both sides of the above equation yields $\frac{1}{2}X_n - \frac{1}{2}X_{n-1} = \frac{1}{2}X_{n+1} - \frac{1}{2}X_n$. This reduces to:

$$X_{n+1} - X_n = X_n - X_{n-1} \quad (3.1)$$

Setting each of these numbers equal to k , results in:

$$X_1 = X_0 + k, X_2 = X_1 + k = X_0 + 2k, \dots, X_{10} = X_0 + 10k$$

Since $X_0 = 0$, this gives: $1 = 0 + 10k$ and $k = 1/10$. Therefore, if A starts with four dollars, the probability he wins the game X_4 is $4/10$. An extension that will illustrate a biased random walk is also provided by Hausner (1971). Here the gamblers have uneven probabilities of winning their bets. Say A has probability p and B has probability q ($q=1-p$). Suppose A and B both start with five dollars. Now, $X_n = pX_{n+1} + qX_{n-1}$, for $0 < n < 10$. To solve these equations, subtract pX_n from each side of the above. This yields:

$$X_n - pX_n = p(X_{n+1} - X_n) + qX_{n-1}$$

$$qX_n = p(X_{n+1} - X_n) + qX_{n-1}$$

$$q(X_n - X_{n-1}) = p(X_{n+1} - X_n)$$

$$X_{n+1} - X_n = q/p (X_n - X_{n-1}) \quad 0 < n < 10 \quad (3.2)$$

Letting $r = q/p$, $X_1 - X_0 = a$, and taking $n=1, 2, \dots, 9$ in equation 3.2, gives: $X_k - X_0 = a(1+r+\dots+r^{k-1}) = a\{1-r^k/1-r\}$.

With $X_0 = 0$ and $X_{10} = 1$, this reduces to: $a = \{1-r/1-r^{10}\}$.

Finally, substitution yields:

$$X_k = \{1-r^k/1-r^{10}\}, \quad r = q/p \quad (3.3)$$

If we let $k=5$ and $p=.6$ for gambler A, Table 3.3 shows how the random walk is biased by A's greater probability of winning. Here, $X_5 = (1-r^5)/(1-r^{10})$.

Table 3.3

A Biased Random Walk

p	.1	.2	.3	.4	.5	.6	.7	.8	.9
r	9	4	7/3	3/2	1	2/3	3/7	1/4	1/9
X_5	.000	.001	.014	.116	.500	.884	.986	.999	.999

Therefore, with $p > .5$ there is a much greater tendency to terminate by winning all the dollars from the opposing gambler and a much smaller tendency to lose all the money. Eisen (1969) reaches the same conclusion. "Thus a skillful gambler, even with a small capital, stands less chance of being ruined than a less skillful gambler with a large amount of capital." Related to simulated annealing, there is a greater likelihood of transitioning to lower values of the objective function (minimization problem) at lower values of the control parameter. The procedure accepts all of the transitions to lower values and probabilistically accepts transitions to higher values, depending on the level of the control parameter. Since these transition probabilities are not equal (1 versus $p < 1$) the result is a biased random walk. As we reduce the value of the control

parameter, the probability of accepting worse configurations is lowered and we get more bias in the random walk.

As a summary, in this section we began by discussing the weighting function used by Metropolis that enables his canonical distribution, V_i distributed $\exp(E_i/T)$, to be approached. This relates the number of configurations to the energy level of the configuration and the temperature parameter. The statistic entropy is used to derive an unbiased estimate of the probability distribution of configurations for combinatorial optimization problems. The resulting Boltzman distribution, characterizing thermal equilibrium, is used to obtain an equation for the average energy level relative to the temperature parameter T . As T is lowered, the probability of accepting worse configurations is lowered and the result is a biased random walk that favors lower energy values. Finally, by gradually lowering the value of T , maintaining thermal equilibrium throughout the process, the average energy can be lowered to its minimum value.

.

IV. SIMULATED ANNEALING'S SUITABILITY FOR INDUSTRIAL ENGINEERING

Although there are many applications of simulated annealing, few are found in industrial engineering. This chapter examines the hypothesis that simulated annealing can be modified to address the requirements of typical industrial engineering problems. More specifically, the real time needs of a dynamic shop floor environment are addressed. The strengths of the algorithm are its flexibility, ease of implementation, and ability to find good solutions to large problems. Its weakness is the large computational time needed to converge to optimality. Therefore, the premise is that simulated annealing can be modified to find near-optimal solutions to various industrial engineering problems.

Simulated annealing is a general purpose heuristic approach to solving optimization problems. There has been a great deal of interest in this approach following the articles of Kirkpatrick et al. (1983) and Cerny (1985). The Science article by Kirkpatrick, with its wide circulation, focused on the potential uses of simulated annealing. Many diverse fields, ranging from mathematics to computer science, soon showed interest in this approach.

The name of the algorithm shows the analogy between solving optimization problems and the physical annealing of a solid, originally introduced by Metropolis et al. (1953).

Here, the objective function (minimization) corresponds to the energy (E) of the solid. The system starts in an initial configuration and gets a random perturbation within the neighborhood of the original configuration. The change is accompanied by a change in the value of the objective function, E . If the change results in a lower value, the new configuration is accepted as the new starting point. If the change increases the objective function value, then the new configuration is accepted probabilistically. The acceptance function usually takes the form of $\exp(-\Delta E/T)$, where T is the control parameter, analogous to temperature in annealing. This provides a mechanism for escaping local minimas. In the implementation of the algorithm, a cooling schedule must be specified. This is used to control the rate of decrease in the control parameter, the number of permutations at each level of the control parameter, the initial value of the control parameter, and the stopping criteria. These are discussed below.

Figure 3.1 gave a very cursory description of the simulated annealing algorithm. In the following section we examine the details of this approach and show the seemingly endless variations available for its implementation.

A. Features of the Algorithm

A brief description is given of the algorithm's terminology (relating to the physical annealing process), its structure, and the various control parameters it uses.

1. Terminology

Table 4.1 below shows the annealing analogy to combinatorial problems and specific examples for a linear programming formulation and the TSP.

Table 4.1

Simulated Annealing Analogy

Physical Annealing	Combinatorial Optimization	LP	TSP
atoms of the solid	variables	X	cities
description of atomic structure	attribute vector or configuration	c	route
energy, E	objective function value	Z	distance of a route
average energy, $\langle E \rangle$	mean value of objective function	$\langle Z \rangle$	average distance
average energy at specific temperature $\langle E_T \rangle$	mean value of objective function at control parameter setting	$\langle Z_T \rangle$	average distance at control parameter setting
thermal equilibrium at temperature setting	current solution at control parameter setting - within epsilon of the average value	Z_T	distance within epsilon of average dist. at T setting
ground state, E_0 , lowest energy state	global optimal configuration	Z^*	globally optimal route

The atoms in the physical annealing process are similar to the variables determining the number of degrees of freedom in a configuration. They determine what energy levels or solutions are attainable. The description of the atoms include their position in a confined space and their momentum. Similarly, a configuration is represented by an

attribute vector describing the variables. In a linear programming problem this is the set of objective function coefficients. Energy is a performance measure that fully describes the atomic structure. More than one structure can have the same value of energy. In the combinatorial optimization problem, this would simply be the value of the function we are attempting to optimize. A linear programming formulation assigns Z to represent this value. As we control the annealing process by reducing the temperature we get lower values of energy as the solid cools. Similarly, as we reduce the value of our equivalent control parameter for combinatorial optimization problems, we move in the direction of lower values of our objective function. Equilibrium is a concept that refers to a steady state relative to a particular temperature setting. This is represented by a value near the average objective function value at a particular setting for the control parameter. Finally, the lowest energy state is equivalent to our globally optimal value.

2. A General Structure for the Algorithm

The following pseudo code is a general outline of the flow for typical simulated annealing algorithms. It is intentionally unspecific to allow future references to tailor it to particular applications.

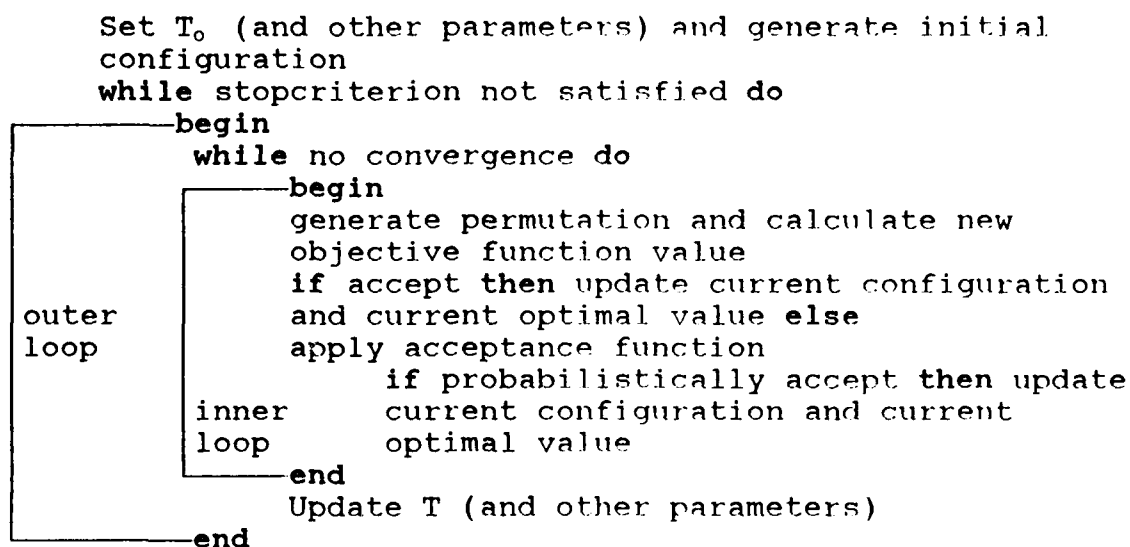


Figure 4.1 A general structure for the simulated annealing algorithm.

The two loops determine the running time and performance of the algorithm. Control of the loops is maintained by the "other parameter" settings. The inner loop is repeated until there is reasonable confidence that an equilibrium value has been reached at the current setting of the control parameter. The outer loop is repeated until there is reasonable confidence that a globally optimal value or near optimal value has been reached. There are many different schemes or **cooling schedules** adopted for various problems that claim to be the most efficient. Many schedules perform well, but require much CPU time when asymptotically reaching the optimum. The difficulty lies in establishing criteria for ensuring this "relative confidence." This process requires a large number of computer operations.

B. Requirements of a Dynamic Shop Floor Environment

Dynamic shop floor environments do not provide enough time for simulated annealing to converge to optimality on large problems. By the time a solution is returned, the decision has been made and implemented or characteristics of the shop floor have changed. Therefore, there is a great need to build flexibility into the algorithm if it is to be of any value on a shop floor. To be useful, the algorithm must be tailored to the problem and run as a heuristic, sacrificing a degree of optimality in return for faster solutions. The algorithm must be flexible enough to allow for the inclusion of raw data from the shop floor. If a machine is operating at less than normal efficiency, its new processing time must be fed back into the algorithm, as this impacts the final schedule. Similarly, if a machine breaks down the operator must be able to suspend and reinitialize the algorithm. The algorithm must allow for the inclusion of constraints that prevent unacceptable schedules. If all jobs must be processed on machine five prior to machine eight, then there must be a mechanism built into the algorithm that prevents schedules that violate this order. To be used by inexperienced operators, a front end expert system must set the various parameters for the efficient running of the algorithm. Results are very sensitive to initial parameter values.

Chapter V presents real time modifications that address the above shop floor requirements.

C. Ways to Modify the Algorithm

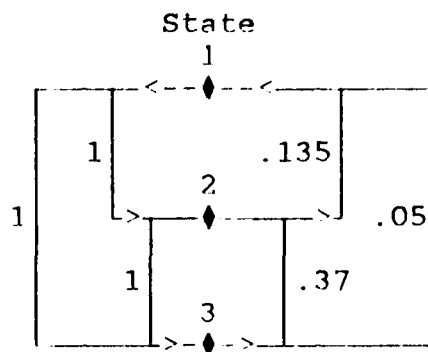
As stated previously, the heart of the algorithm is its probability of accepting transitions away from optimality. This is what allows the algorithm to escape local minimas. Therefore, the area representing the greatest potential for change is the basic probability of acceptance. Other modifications ultimately impact this probability as well. An example using a Markov Chain is used to illustrate this.

1. An Example Using Markov Chains

Another method of characterizing the behavior of the algorithm is through the use of Markov Chains. The following example shows a simple three state system. Ω_i represents the number of configurations in state i , while E_i is the energy or objective function value for state i (a minimization problem). The state diagram is shown below.

3 STATE SYSTEM:Probability of Acceptance

$$\begin{array}{ll}
 \Omega_1 = 6 & E_1 = 4 \\
 \Omega_2 = 4 & E_2 = 2 \\
 \Omega_3 = 1 & E_3 = 1
 \end{array}$$



$$e^{(E_1 - E_2)/T}, \quad T=1$$

A transition from one configuration to another occurs after the new configuration is generated from the current one and then it is accepted. Therefore, generation and acceptance matrices are needed to obtain the transition matrix. These are shown in the following example.

Steady State Markov Chain

<u>Generation</u>				<u>Acceptance (T=.5)</u>				<u>Transition</u>			
i\j	1	2	3		1	2	3		1	2	3
1	.5	.4	.1	1	1	1	1	1	.5	.4	.1
2	.6	.3	.1	2	.018	1	1	2	.011	.889	.1
3	.6	.4	0	3	.0025	.135	1	3	.0015	.054	.945

$$\left\{ e^{(F(i) - F(j))/T} \right\}$$

(Metropolis Criterion)

The acceptance matrix is non-stochastic since all moves towards optimality are accepted and moves away from optimality are probabilistically accepted. The following notation is used, where C_k denotes the control parameter.

$$\text{Generation Probability} = G_{ij}(C_k)$$

$$\text{Acceptance Probability} = A_{ij}(C_k) = \begin{cases} 1 & , f(j) \leq f(i) \\ e^{(f(i) - f(j))/C_k} & , \text{o.w.} \end{cases}$$

$$\text{Transition Probability} = T_{ij}(C_k) = \begin{cases} G_{ij}(C_k)A_{ij}(C_k) & , i <> j \\ 1 - \sum_{j=1}^3 T_{ij}(C_k) & , i=j \end{cases}$$

To determine the transition probability, simply multiply the corresponding off-diagonal elements of the generation and acceptance matrices. Diagonal elements are then found by subtracting the sum of off-diagonal elements from one. The greatest potential for modifying the algorithm lies in the

probability of accepting worse configurations, the lower triangular portion of the acceptance matrix.

For T=.5:

Boltzman at equilibrium:

Steady State M.C.:

$$P_1 \{x=1\} = \frac{e^{-4/.5}}{Z(T)} = \frac{.0003}{.1533} = .002$$

$$P\{x=1\} = .01$$

$$P_2 \{x=2\} = \frac{.018}{.1533} = .12$$

$$P\{x=2\} = .35$$

$$P_3 \{x=3\} = \frac{.135}{.1533} = .88$$

$$P\{x=3\} = .64$$

This example is also used to compare the Boltzman distribution (characterizing thermal equilibrium) to results for a steady state Markov Chain. It is seen that at high values of the control parameter the Boltzman somewhat approaches the steady state Markov Chain. It must be pointed out that this simple example, containing only three states, was not chosen for its good results, but rather to illustrate the performance of the algorithm. Reducing the control parameter gives a much closer approximation to steady state.

<u>Generation</u>				<u>Acceptance (T=.25)</u>				<u>Transition</u>			
	1	2	3		1	2	3		1	2	3
1	.5	.4	.1	1	1	1	1	1	.5	.4	.1
2	.6	.3	.1	2	.0003	1	1	2	.00018	.89	.1
3	.6	.4	0	3	0	.018	1	3	.0000037	.0072	.993

For $T=.25$:

Boltzman at equilibrium:

Steady State M.C.:

$$P_1 \{x=1\} = \frac{e^{-4 / .25}}{Z(T)} = \frac{.000}{.01834} = 0$$

$$P\{x=1\} = 0$$

$$P_2 \{x=2\} = \frac{.00034}{.01834} = .02$$

$$P\{x=2\} = .06$$

$$P_3 \{x=3\} = \frac{.018}{.01834} = .98$$

$$P\{x=3\} = .94$$

As the control parameter is further reduced (.125), the Boltzman distribution is equivalent to the steady state Markov Chain. Therefore, the asymptotic, steady state behavior of the algorithm is accurately represented by the Boltzman distribution. Romeo and Sangiovanni-Vincentelli (1985) formally prove that the algorithm will produce the optimal solution when it is run long enough. They first show that the Markov chain is irreducible, aperiodic, and recurrent. Thus, a stationary probability distribution does exist. The form of the limiting distribution is then examined as the control parameter approaches zero. The

steady state probability for each optimal solution is equal to:

$1/(\text{the number of optimal solutions})$, while all other states have a probability of zero. Finally, they give conditions regarding the generation and acceptance probabilities that ensure the steady state results above. The functions specified by simulated annealing satisfy such conditions.

Although it is reassuring to know that the global optimum is guaranteed under certain conditions, the real time aspects of a shop floor environment will not permit these conditions. The modifications presented in the next chapter address the real time needs of the shop floor. They basically alter the probability of accepting transitions to worse configurations, the lower triangular portion of the acceptance matrix. The modification with the greatest impact is the probability of acceptance.

V. A MODIFIED SIMULATED ANNEALING FORMULATION

A. The General Structure

This section uses the material in Chapter IV as a foundation to present the modified formulation. The modified approach simply gives the user some additional capabilities that address real time needs. Specifically, the ability to accept shop floor data and interactively adjust the algorithm's parameters to reflect these updates. A surge of data will often cause a previously good solution to be inappropriate. Adjusting the parameters to run the algorithm longer allows a user to find accurate solutions that reflect current system status. The use of alternate acceptance functions gives the user some added flexibility. He can make a direct tradeoff between solution quality and processing time. With more decision time available, the standard acceptance function is employed. The search slowly migrates to better solutions and more search time is spent in lower regions of the configuration space. When a decision is needed soon, an acceptance function such as JS2 is used to quickly find acceptable solutions at the risk of getting trapped in a suboptimal local minimum. This feature capitalizes on the main purpose of all heuristics - trading off a degree of optimality for processing time. With this modification each user can make an individual tradeoff to reflect current conditions surrounding the process. The expert system relieves an inexperienced user from having to

select an efficient set of parameters for a given size problem and an allotted amount of processing time. Finally, the constraints module prevents unacceptable solutions. The result of these modifications is an algorithm that is tailored to the particular needs of each user. An operator in one situation selects a set of parameters, constraints, and an acceptance function that yields a solution when it is needed. Another operator in a different situation tailors the algorithm for his needs to give a different solution when it is needed. The modifications provide a means for each user to bias the random walk in a way that addresses their particular needs.

Figure 5.1 shows pseudo code for the modified simulated annealing formulation.

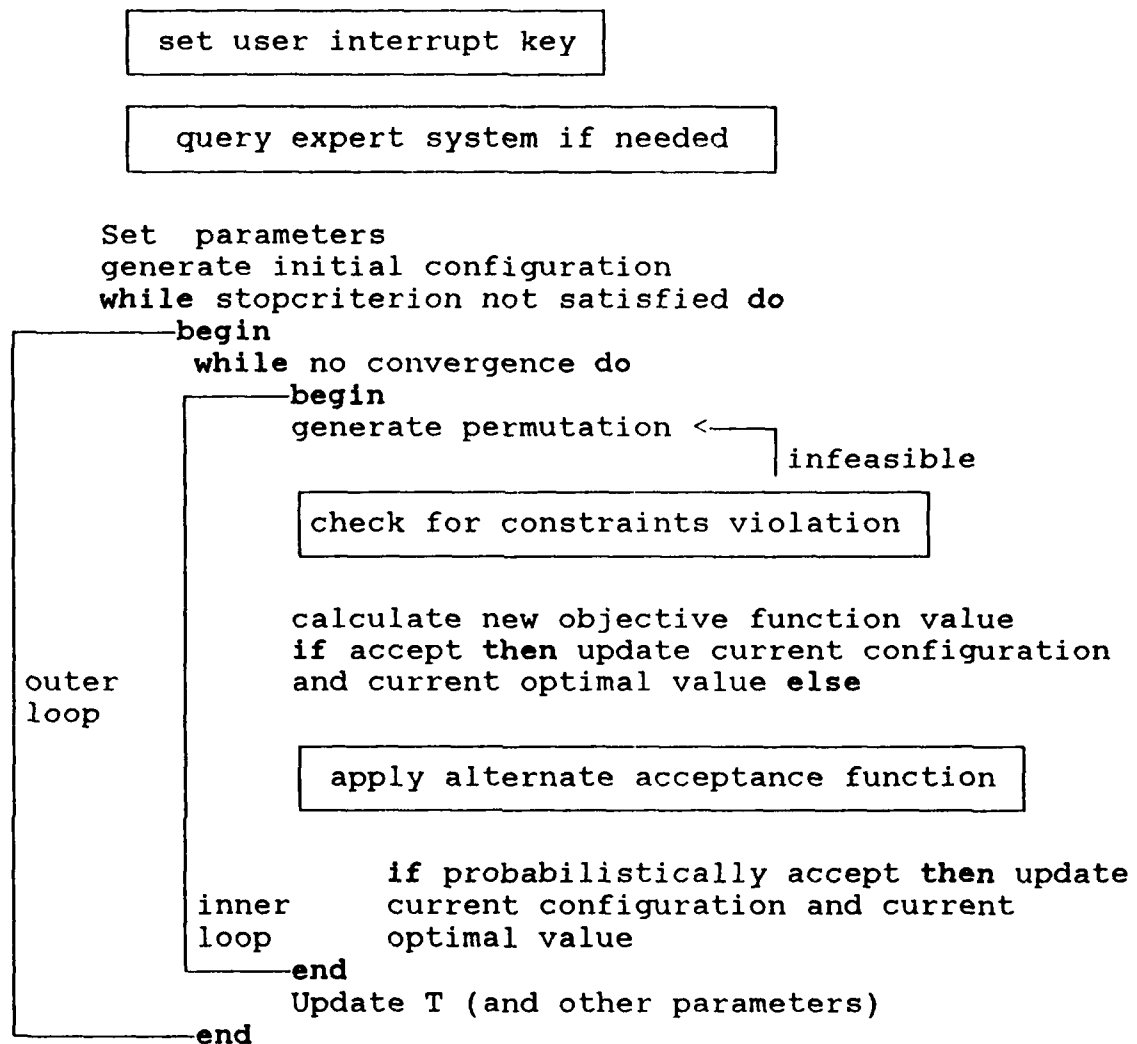


Figure 5.1 A modified structure for the simulated annealing algorithm.

The modified algorithm begins by establishing the user interrupt key. At any time during the running of the algorithm this key can be activated, causing the algorithm to be suspended. A subroutine is then accessed, allowing the user to reinitialize parameter settings to respond to changes on the shop floor (a surge of data). Execution is

resumed at the point where the interrupt occurred. Next, the front end expert system is provided for the inexperienced user. If queried, this module simply asks the user for the size of the problem (number of jobs and machines), the amount of processing time available, and the frequency of shop floor data. With a variety of processors to choose from, it would also ask for the type of processor. Based on the user's response, the expert system selects a set of parameters that will run efficiently the algorithm and provide a quality solution when it is needed.

Contrary to most heuristics (branch and bound, dynamic programming) simulated annealing is flexible enough to allow the user to build in constraints that prevent infeasible solutions. If there are constraints that define acceptable solutions, the constraints module is checked immediately after the permutation is made. This prevents the unnecessary calculation of an objective function value for unacceptable solutions. Once an acceptable solution is found, the standard flow is resumed and the two configurations (current one and trial one) are compared. The last modification simply replaces the acceptance function $e^{(f(i) - f(j))/T}$ by an alternative function. Here, the aim is to arrive at a solution faster. The rest of the algorithm remains unchanged, with the same inner and outer loop construction.

Figure 5.2 displays the four modifications and their major area of impact on the algorithm's performance. These impact areas are the points where bias is being introduced to the random walk.

Modification	Primary Impact on Traditional Form
1. Constraints Module	Generation of Initial Configuration Permutation Mechanism
2. Expert System	Cooling Schedule Temperature Parameter
3. User Interrupt	Cooling Schedule
4. Acceptance Function	Acceptance Function Cooling Schedule

Figure 5.2 Modification impacts.

Figure 5.2 depicts the modifications to the standard use of simulated annealing. Appendix G shows the code with the inclusion of these modifications.

The principle modification is the use of alternate acceptance functions. Again, these allow the user to exploit the heuristic by trading off a degree of optimality for computational time. We must examine the probability of acceptance when attempting to explain the differences in performance for the two versions of simulated annealing.

Since their starting configurations and permutations are identical, the only difference is their acceptance function. This function defines the shape of the cooling schedule. The acceptance functions for the standard form and the uniform version are shown below, where X is the probability of acceptance.

$$\text{SA-NE: } X = \exp\{(D-TD)/T\}$$

$$\text{SA-UN: } X = 1 - \{\alpha*(TD-D/T)\}$$

This function defines the shape of the cooling schedule. Figures 5.3 and 5.4 show the curves for the standard form and inverted uniform version respectively.

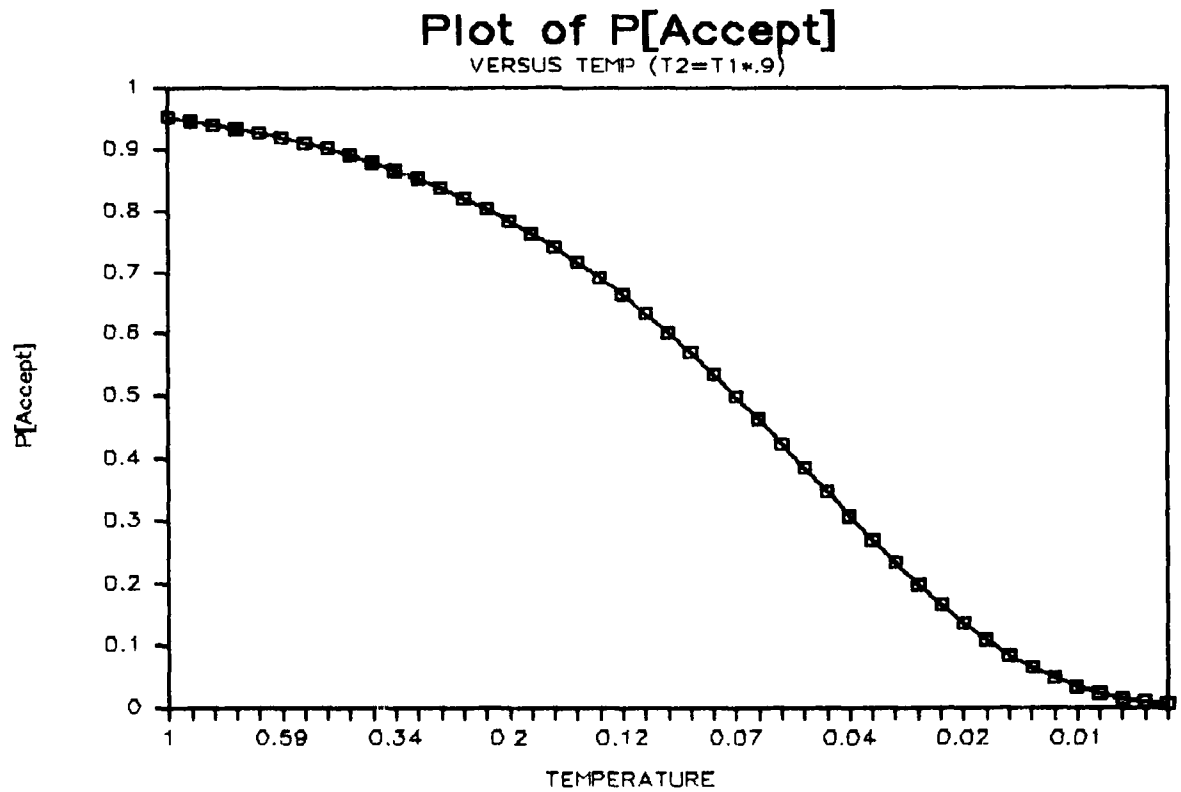


Figure 5.3 Probability of accepting worse configurations versus control parameter (for standard form acceptance function and fixed delta cost).

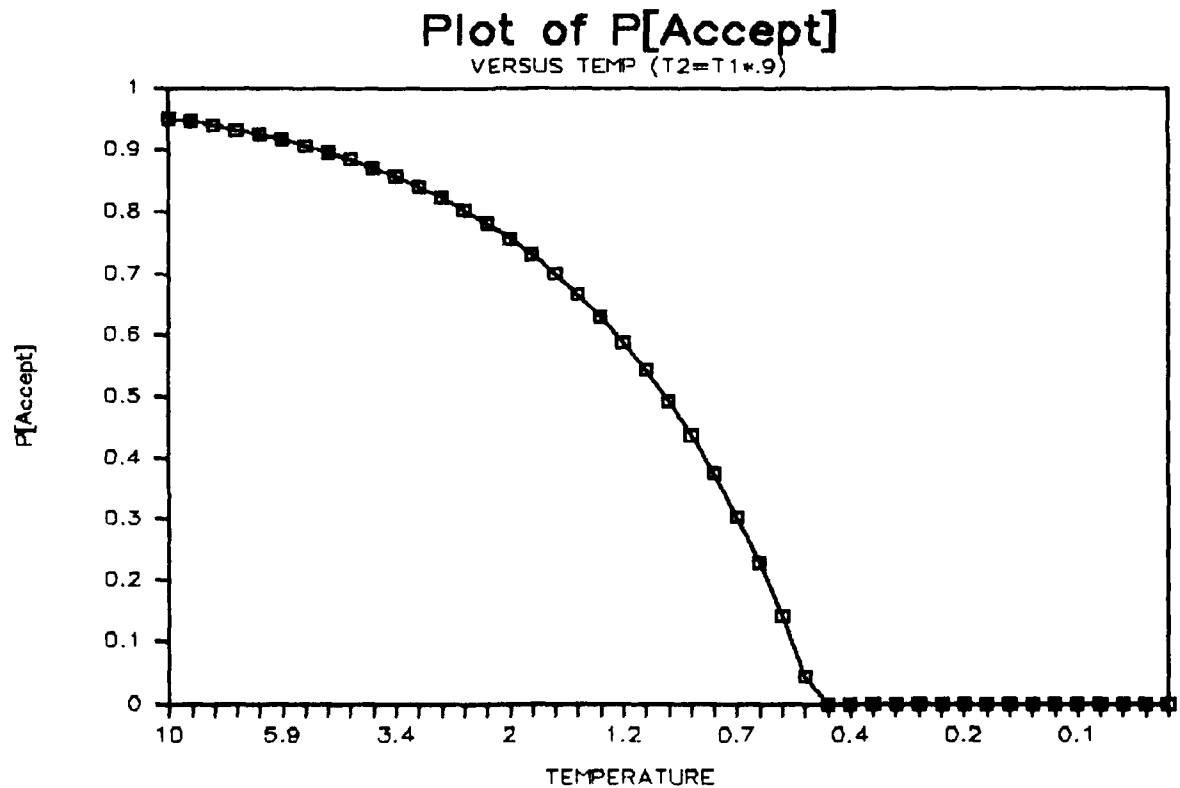


Figure 5.4 Probability of accepting worse configurations versus control parameter (for pseudouniform acceptance function and fixed delta cost).

As depicted, the curves are initially very similar. Asymptotically, however, the inverted uniform becomes essentially a greedy heuristic only accepting better configurations. The relatively good performance (see Table 7.4) of this version prompted the creation of two additional family of acceptance functions. Curves for FS1 and FS2 below are displayed in figures 5.5 and 5.6 respectively.

$$\text{FS1: } X = \beta * (T/|D - T|)$$

$$\text{FS2: } X = T * (D/|D - T|)$$

Here, X represents the probability of accepting the worse configuration. T is the control parameter, D is the current solution, $|D - T|$ is the trial solution, and β is input by the user as a weighting function. Large values of β (FS1) weight T more and the heuristic behaves more like a local search, finding quickly the local optimum, but risking suboptimality. This acceptance function performs better on configuration spaces containing many low lying local minimas.

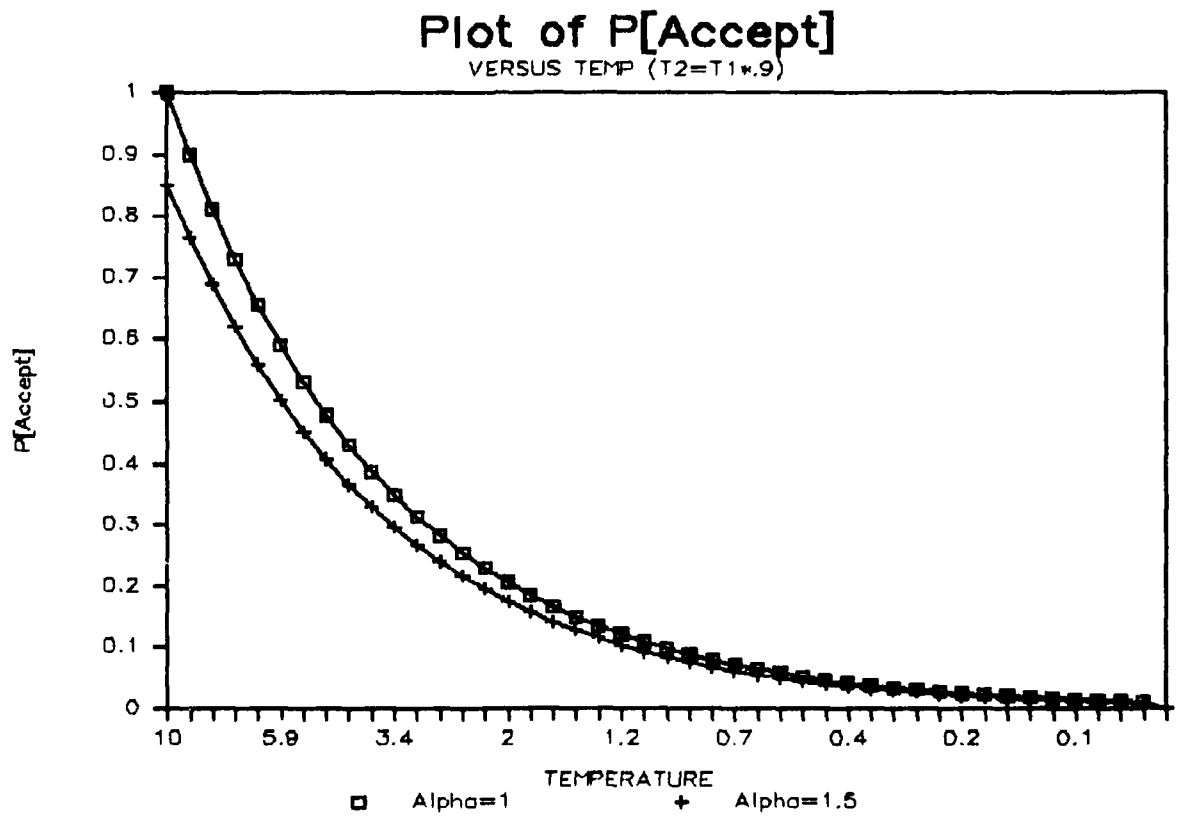


Figure 5.5 FS1 probability of acceptance.

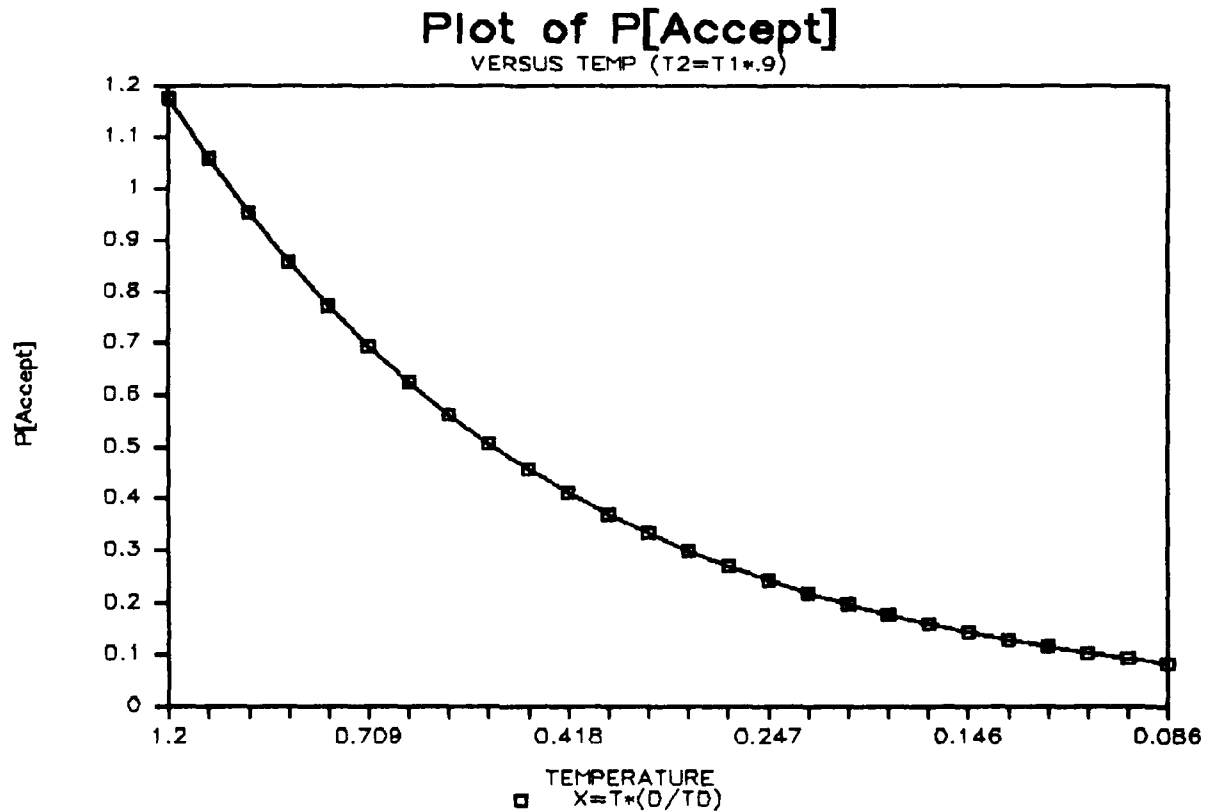


Figure 5.6 FS2 probability of acceptance.

This function is a weighting of the relative ratio of the objective function values (D/TD). The control parameter T is used as the weighting factor.

Table 7.5 of chapter VII gives results for the alternate acceptance functions tested.

1. A Modified TSP Formulation

The attached TSP code provides some additional capabilities to support an operational application. This program graphically displays the performance of the

algorithm as it searches for the optimal route. Current global optimums are recorded and the F10 function key allows the operator to briefly interrupt the algorithm and display the current optimal solution. The four modifications previously discussed are implemented and reflected in Figure 5.7. There is also a stopping criterion based on the amount of available computer time prior to the decision point.

BEGIN

INITIALIZE

SET USER INTERRUPT KEY

CONSULT EXPERT SYSTEM AND INITIALIZE PARAMETERS

Generate Route ($R_0(N)$)

CHECK CONSTRAINTS MODULE

Calculate Distance (D_0)

Set Temp Decrement (α) and Repetitions Increment (β)

Set Number of Trials Counter (NT) 'stop criterion

REPEAT

For $L=1$ to L_k

Begin ' Permute Temporary Route ($TR(N)$)

Generate 2 Random Integers(1-N), J and K

Reverse Direction Between cities J and K

CHECK CONSTRAINTS - IF INFEASIBLE GOTO BEGIN

Calculate Distance for Temporary Route (TD)

If $TD \leq D$ Then UPDATE Else

CALCULATE $P[A]$ ' PROBABILITY OF ACCEPTANCE

If $P[A] > \text{Random}[0,1)$ Then UPDATE

End

$L_k = L_k * \beta$

$T = T * \alpha$

UNTIL Stopcriterion (NT) is met

END

```

UPDATE                      ' Reset Current Route and Distance

    R(N) = TR(N)

    D = TD

RETURN

```

Figure 5.7 Pseudo code for the modified simulated annealing algorithm applied to the TSP.

B. Using Simulated Annealing in an Applied Setting

The majority of the literature is focussed on finding a set of parameters that allows the algorithm to find efficiently the global optimum. Although this is useful for understanding the performance of the algorithm, in practice too much time is spent asymptotically converging to the optimum. The strength of this heuristic is its simplicity, flexibility, and ability to quickly find a good solution. We will attempt to capitalize on the algorithm's strong features when implementing it in an applied setting.

The attached program code for the TSP problem provides the user with a "hot" key. The F10 function key is used to interrupt briefly the algorithm and print the current best solution found. This requires the use of an additional variable to store and update the globally optimal solution. Used this way, an operator can set up the program input and let it run for a variable length of time. Whenever a solution is needed the algorithm can be interrupted. With

the low cost of small computers there can be a dedicated processor running the simulated annealing algorithm non-stop while users query the system on an as needed basis.

In addition to the hot key, a graphical depiction of the algorithm's performance is displayed to the user. This provides user feedback regarding the current status of the algorithm. If the search is in an area far from the global minimum the operator immediately uses the best solution found. If the search is near the global minimum and seems to be finding new minimums the operator is free to trade off processing time for an improved solution. With the additional information provided by the graph, the algorithm can take advantage of operator judgement and becomes a more useful tool.

C. Parameter Settings

The common approach to simulated annealing is to start off with a high temperature setting, allowing many transitions to higher cost configurations, and gradually lower the temperature parameter until it approaches zero. This procedure is governed by the cooling schedule. The parameters that determine this cooling are:

- (1) the initial value for the temperature, c_0
(the control parameter),
- (2) the function for decrementing the temperature,
- (3) the number of trials at each temperature
setting, referred to as individual Markov
chains, and
- (4) the stopping criterion.

Referring to Figure 5.7, the initial temperature is T_0 and the Markov chains are of length L_n . In addition to the cooling schedule, we must also specify the initial configuration, i_{start} .

D. Cooling Schedules Employed

Collins, Eglese, and Golden (1988) have listed a number of different schemes for (2) through (4) above. Some of these are shown below:

(2) Temperature Decrement, $T(t)$

Constant	$T(t) = C$
Arithmetic	$T(t) = T(t-1) - C$
Geometric	$T(t) = \alpha(t)T(t-1)$
Inverse	$T(t) = C/(1+\delta t)$
Logarithmic	$T(t) = C/(\ln(1+t))$

(3) Number of Trials, $N(t)$

Single	$N(t) = 1$
Constant	$N(t) = C$
Arithmetic	$N(t) = N(t-1) + C$
Geometric	$N(t) = N(t-1)/\alpha(t)$
Logarithmic	$N(t) = C/\log(T(t))$
Energy	Repeat until average energy is changing little
Acceptances	Repeat until a number of acceptances have occurred
Rejections	Repeat until a number of rejections have occurred

(4) Stopping Criterion

Iterations	Fixed number of iterations
Temperature	$T(t) \leq$ final value of T
Energy	$\langle E \rangle$ is changing little at successive temperatures
Acceptance	Few acceptances are occurring at successive temperatures

Aarts and Van Laarhoven (1988) propose a cooling schedule that allows the annealing process to maintain a

state of quasi equilibrium at each temperature setting. This is a requirement for reaching a global optimum. In general, the temperature must start off at a relatively high setting to avoid getting trapped in local optimums. At each update of the temperature parameter the number of trials is increased. Say we are trying to minimize a function. As we get lower values of our objective function there will be proportionately more trial configurations to choose from with higher objective function values. To obtain enough accepted transitions to allow an estimate of the average equilibrium value, we must generate a greater number of trial configurations.

E. Formulation of a Problem

The remainder of this chapter relates previous theoretical material to an industrial engineering application. The problem involves scheduling jobs on machines such that the makespan is minimized.

Although the formulation of a problem is fairly straightforward, generating configurations is sometimes non-trivial. The various parameter settings allows for a great deal of variation in the algorithm's performance and requires some sophistication on the part of the user.

1. Generating the Initial Configuration

Since the algorithm is independent of the initial configuration (Kirkpatrick, Gelatt, & Vecchi, 1983), it will eventually converge from any point. That is, providing it

uses a cooling schedule that guarantees optimality. Some random selection of starting values is usually recommended (Catthoor, de Man, & Vandewalle, 1988). In practice however, we are seldom afforded the necessary time to reach optimality. For this example, the starting configuration is the initial schedule. The mechanics of generating the configuration ranges from being simple to being complex. It can require some creative insights into the problem at hand. An example is presented in the formulation of the minimum cut problem.

2. Generating A Permutation

A permutation is a configuration or vector that is in a neighborhood of the current configuration. By neighborhood, we mean that some simple procedure maps one vector into the other. To guarantee convergence, permutations must allow all configurations to be reached from all other configurations. In terms of a Markov Chain, there is some probability greater than zero that we can transition from one configuration to the other. Here, all states communicate and the Markov chain is said to be irreducible (Hillier & Lieberman, 1980). Since permutations are minor modifications, one vector's solution will not differ greatly from the other's. Two types of permutations are swapping and reversal. Examples of these common permutations are given in Figures 5.8 and 5.9 for zero-one programming and TSP problems respectively.

In the first example there are two sets. Those with coefficients equal to zero and those with coefficients equal to one. A variable is chosen at random from one set and another from the opposite set. The assignments to their coefficients are then swapped. The current configuration for a 10 variable problem might be the vector $(0,0,1,1,0,1,1,1,0,0)$. Permuting this by randomly selecting x_2 and x_3 we get the vector $(0,1,0,1,0,1,1,1,0,0)$. The objective function value is only slightly different for the new configuration. For a maximization problem, it changes by an amount $(c_2 - c_3)$, where c_i is the objective function coefficient for variable x_i . This is illustrated in Figure 5.8.

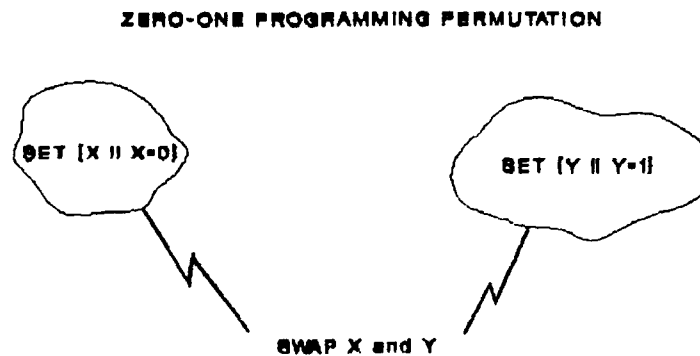


Figure 5.8 Swapping permutation for zero-one programming problem.

A similar permutation for the TSP reverses the direction of travel between two randomly selected nodes. A current configuration that traverses five nodes in the order (1-2-5-3-4) is transformed into a new configuration (1-4-3-5-2). Here, the second and fifth node along the route are selected as endpoints and the path between them is reversed. For most TSPs, the new route will only differ slightly from the current route. The illustration is provided in Figure 5.9.

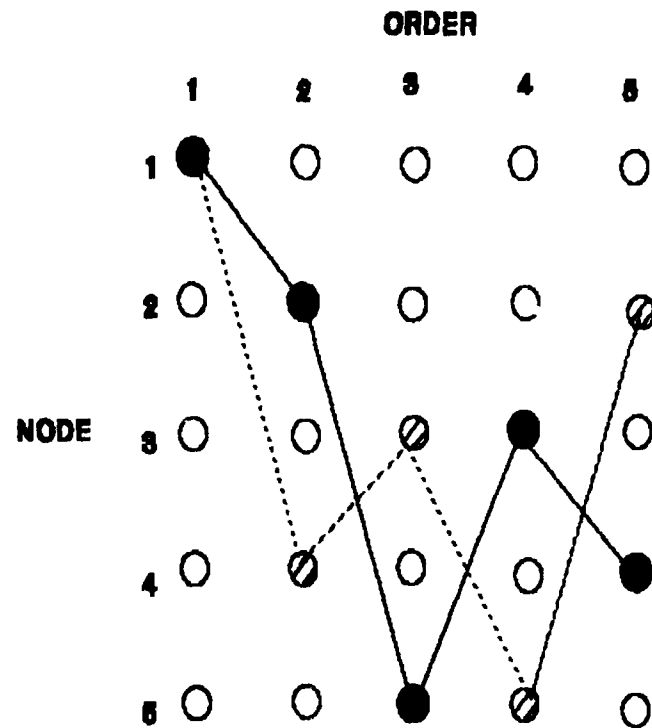


Figure 5.9 Reversal permutation for the traveling salesmen problem.

An alternative permutation is obtained by simply switching the position of two randomly selected nodes. This

results in a route whose path can be much larger than the current route. Therefore, the new route is not in the neighborhood of the current route and this scheme is not recommended.

For the shop floor formulation, the swap permutation is used. Two randomly selected processes (assignment of a job to a machine) interchange positions in the overall sequence. The new makespan is slightly different.

3. The Concept of Energy

Energy in the physical annealing process is a measure used to represent the configuration of atoms in the solid. In an optimization problem energy is the solution or value of the objective function for a given assignment to the variables. For example, in the job shop scheduler it is the makespan for a given schedule. High energy states equate to sub-optimal solutions and low energy states equate to near optimal solutions. The lowest energy state is referred to as the ground state. In optimization problems this is called the globally optimal solution.

4. The Temperature Parameter

Just as temperature is used in annealing to lower the energy state of the solid, it is used in optimization to move in the direction of better solutions. The units of the temperature parameter must be in the same units as the objective function. In optimization temperature is a meaningless term and is replaced by the term control

parameter. The purpose of this parameter is to control the probability of accepting transitions to "worse" configurations. Initially, it is set high to allow transitions to less optimal configurations. This allows the algorithm to move out of local optimals. As the global optimum is being approached, the control parameter is reduced and fewer sub optimal moves are allowed. When the parameter is set to zero, the algorithm behaves like a greedy heuristic, allowing only those moves that improve the objective function value.

The actual probability of accepting a worse configuration is a function of the amount of degradation in the solution and the value of the control parameter. This forms a ratio as depicted by $(-\delta C/T)$. Here, $-\delta C$ is the amount of degradation in the solution and T is the value of the control parameter. Figures 5.10 and 5.11 show the relation between the probability of accepting sub-optimal transitions and this ratio.

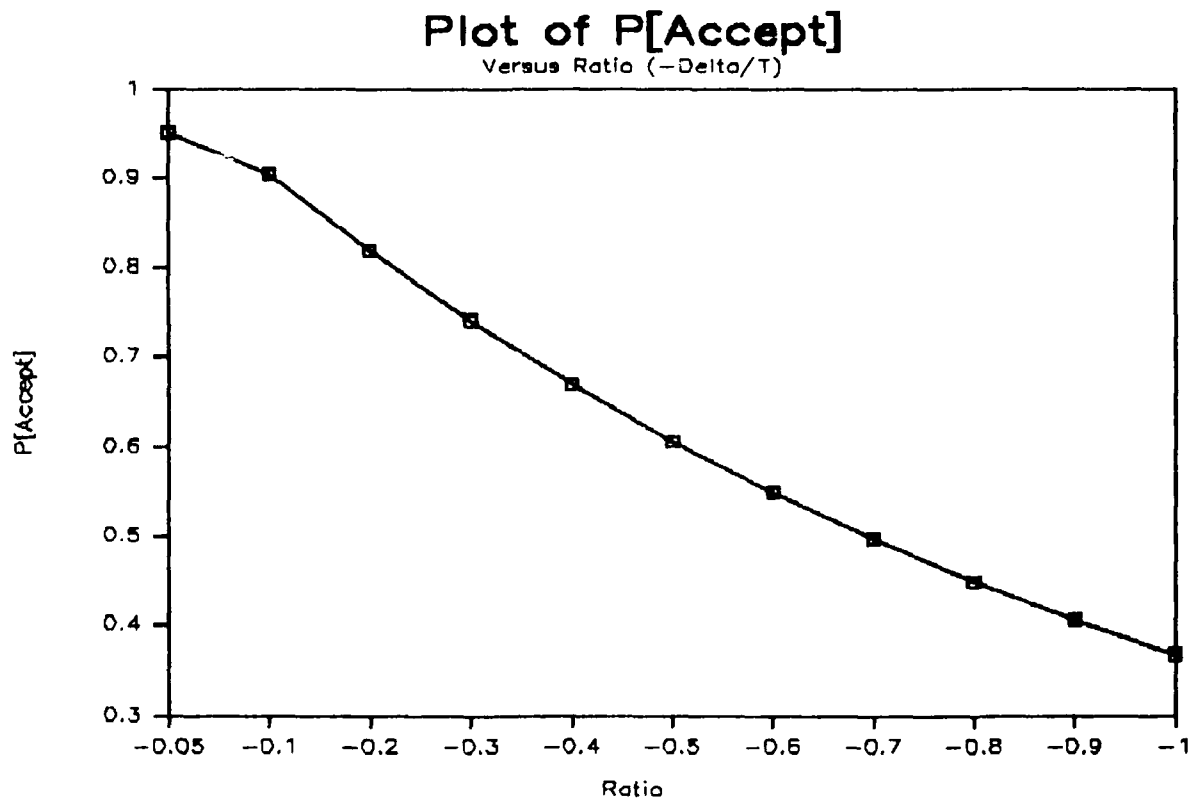


Figure 5.10 Relation between the probability of accepting a sub-optimal configuration and the ratio ($-\delta C/T$) over the range 0 to -1.0.

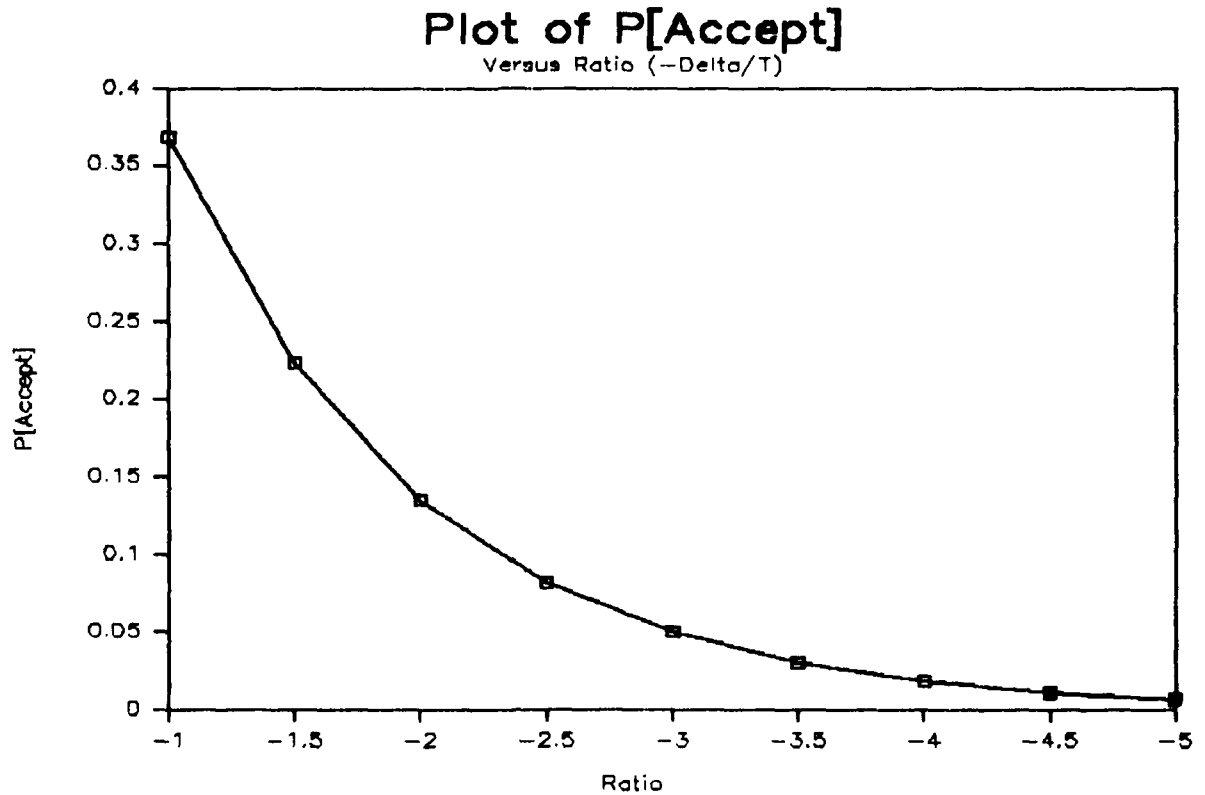


Figure 5.11 Relation between the probability of accepting a sub-optimal configuration and the ratio ($-\delta C/T$) over the range -1.0 to -5.0.

As displayed, large deviations are accepted with the control parameter set relatively high and only small ones are permitted with a low setting.

The initial value for the control parameter must be set relative to the range of values for the objective function. There is a tradeoff between the amount of running time required and the assurance that the algorithm will not get stuck in a local optimum. Aarts and Korst (1985) introduce

the idea of an acceptance ratio equal to the number of accepted transitions divided by the number of attempted transitions. His approach is to continue to update the starting value of the control parameter until this ratio becomes acceptable. Kirkpatrick (1984) equates this process to melting the solid in the physical annealing sense. He implements this approach by doubling the control parameter until the acceptance ratio is greater than 80 percent. This has the drawback of requiring more computer time. An alternative and easily implemented approach is given by Cerny (1985) for the hole drilling problem. He assigns the initial control parameter a value equal to the average distance between holes (nodes of a TSP). A similar approach is used for the shop floor formulation. Here, the initial control parameter is given a value equal to the average processing time of a machine on a job.

5. The Cooling Schedule

The cooling schedule is comprised of two sets of values. First, a set of values for the control parameter. Second, each value of the control parameter has a finite number of transitions. This set of transitions we call a finite length homogeneous Markov chain. Again, in this example it is the number of trial schedules at each value of the control parameter.

Settings for the control parameter include the initial value, the sequence obtained by applying a decrement

function, and the final value. We have discussed the starting value and will discuss the final value (stopping criterion). The decrement function allows the algorithm to gradually lower the probability of accepting transitions to less optimal configurations. A number of different strategies can be employed, depending on the problem at hand. Most of the literature recommends using a geometric function for decrementing the control parameter. This is of the form: $T(t) = \alpha T(t-1)$. In words, the control parameter T at time t is equal to some constant α times the control parameter at time $(t-1)$. The value of α is recommended to be in the range .8 to .99 (Aarts & Korst, 1989). For instance, if we want to find a solution that is very close to optimal and are willing to commit much computer time, we then decrement the parameter very slowly. Here, we use an α equal to .99. If we want a faster solution and are willing to sacrifice some degree of optimality we set α equal to .8. In a situation where we must have a reasonable solution very fast, then we set α much lower, to say .5. Bonomi and Lutton (1984) cautions, however, that a temperature decrease that is too rapid is likely to trap the algorithm in a configuration that is "far" from the optimal one.

The second set that makes up the cooling schedule, the number of trials at each value of the control parameter, is determined by the requirement to maintain quasi equilibrium at each setting. Let L_k denote the length of the K^{th} Markov

chain and c_k the corresponding value of the control parameter. Quasi equilibrium is achieved if $a(L_k, c_k)$, i.e. the probability distribution of the solutions after L_k trials of the K^h Markov chain, is sufficiently close to $q(c_k)$, the stationary distribution at c_k .

$$|a(L_k, c_k) - q(c_k)| < \text{epsilon}$$

Requiring the final value of the Markov chain to be within some small range of the equilibrium or average value (at each c_k) prevents the algorithm from being trapped in local optima. Since transitions are accepted with decreasing probability, as c_k approaches 0, L_k approaches ∞ . Thus, there is a trade-off between large decrements of the control parameter and small Markov chains. Usually, most approaches apply small decrements in c_k to avoid long Markov chains. For practical applications we need to bound the length of the Markov chains for small values of c_k (Aarts & Korst, 1989). Another approach given by Aarts for achieving a polynomial time cooling schedule assumes a condition for quasi equilibrium being:

$$|q(c_k) - q(c_{k+1})| < \text{epsilon}, \quad \text{for each } k \geq 0.$$

At each control parameter setting we are within some small range of the previous equilibrium value.

Much of the work reported on in the literature focuses on finding optimal annealing schedules given a fixed number of Monte Carlo steps. Randelman and Grest (1986) report that much less attention is given to the dependence of the optimal value on the total number of Monte Carlo steps. In their work, they show empirically that cooling slower yields better results.

6. The Acceptance Function

Although the function that is generally used is the negative exponential, provided by Boltzman, there are a number of others that could be employed. We need not be restricted to the physical annealing analogy that is responsible for originating the use of simulated annealing for optimization problems. One of the pioneers of simulated annealing, Cerny (1985), states that the algorithm could be proposed without any reference to statistical physics. Nahar, Sahni, and Shragowitz (1985, 1986) implement a number of alternative acceptance functions. Their contention is that although annealing has a sound theoretical basis in the physical domain, no such basis exists for its application to combinatorial optimization problems. They report good results for optimal linear arrangement problems and mixed results for the TSP. In a real time industrial engineering application, where a solution is needed in a limited amount of time, other acceptance functions find superior solutions.

The three alternative functions previously presented are used in the shop floor formulation.

7. The Stopping Criterion

After the problem is mapped into a simulated annealing space and the algorithm initiated, there must be some mechanism for terminating the algorithm. Again, there is a great deal of flexibility for the stopping criterion. Three of the most common, simple, and intuitively appealing are given below.

- 1) when the control parameter is sufficiently small:

$$c_k \leq \text{epsilon}$$

- 2) when the value of objective function at quasi equilibrium is not changing significantly Romeo (1984):

$$|q(c_k) - q(c_{k-1})| \leq \text{epsilon}$$

- 3) when it becomes very difficult to find a better solution, i.e. the acceptance ratio is very small:

$$\text{the number of trials (without an acceptance)} \geq N$$

The specific parameter setting is a function of the decision makers' attempt to trade-off running time for the degree of optimality.

In summary, to apply the algorithm we need three components. We must have a problem depiction, consisting of an initial configuration (from the solution space) and an expression for calculating the objective function value. For each configuration we must have a permutation mechanism that provides a means of generating a local permutation. Finally, we need a cooling schedule to control the rate of decrease in the control parameter. With these three components we can apply simulated annealing. Tailoring the algorithm to run efficiently for a particular problem requires fine tuning the parameter settings for the cooling schedule. This schedule is comprised of the set of values for the control parameter, c_k , and the number of trials at each value necessary to reach quasi equilibrium. Three parameters are needed to determine the range of values for the control parameter. We must specify the initial control parameter setting, c_0 , the decrement function, and the final control parameter value.

To Apply Simulated:

1. problem depiction
 - an initial configuration
 - an expression for obtaining the objective function value
2. a permutation mechanism
3. a cooling schedule

To Tailor Simulated Annealing:

4. cooling schedule parameters
 - an initial value for the control parameter, c_0
 - a decrement function for the control parameter
 - a final control parameter value
 - a finite number of repetitions (trials) at each value of the control parameter.

The focus of the remainder of the material is on the actual performance of the modified simulated annealing algorithms for the job shop and flow shop problems.

VI. APPLICATION OF THE MODIFIED SIMULATED ANNEALING ALGORITHM TO THE JOB SHOP SCHEDULING PROBLEM

A. Job shop Formulation

The general Metropolis procedure is followed in the formulation of two common industrial engineering problems. The job shop and flow shop problem formulations incorporate the following modifications to the standard use of simulated annealing:

- 1) a front end expert system for inexperienced users,
- 2) a constraints module,
- 3) a user interrupt capability, and
- 4) the use of alternate acceptance functions.

Figure 6.1 shows the schematic of a proposed shop floor scheduling system employing the four changes. The modifications are intended to capitalize on the strengths of the algorithm, while addressing the real time operational needs of a job shop environment. The choice of alternate acceptance functions provides the user a degree of flexibility in trading off solution quality for computational time, which is the purpose of any heuristic.

PROPOSED SHOP-FLOOR SCHEDULER

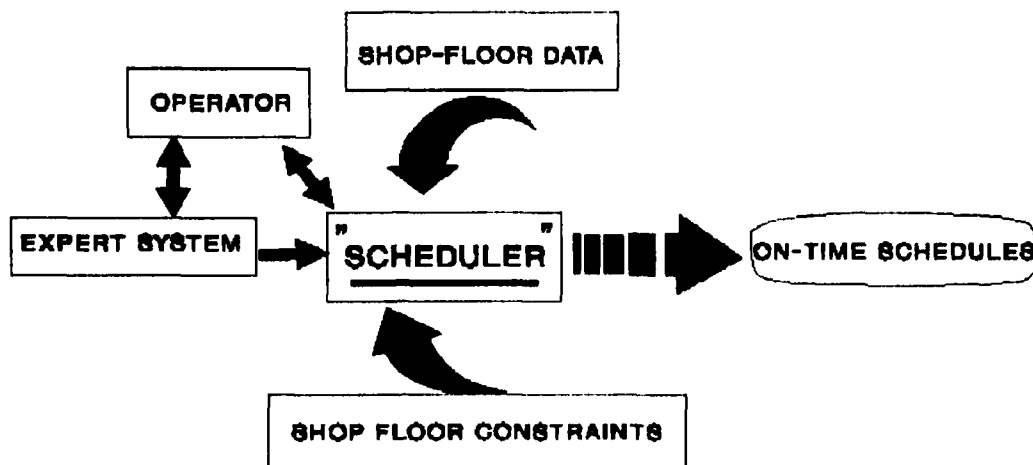


Figure 6.1 Shop floor scheduler.

The optimizer, the main part of this system, is simply the simulated annealing algorithm formulated for the job-shop scheduling problem. The algorithm allows new shop-floor data to be entered. If there is a surge of data that severely degrades the solution the user can interrupt the process through the user interface block. New parameter values are then assigned for the cooling schedule. For a worst case scenario the user can reset the parameters to reinitialize the algorithm. If there are only minor disturbances the user can simply refine the parameters in an

attempt to find quickly a better solution. Inexperienced users have the option of consulting a front end expert system. The system queries the user to find the size of the job, the speed of the computer, and when a schedule is needed. The result is a set of cooling schedule parameters that will efficiently run the algorithm within the limitations defined by the user. The user also has the option of creating a set of constraints. One such constraint might say the schedule must sequence job 3 to be processed on machine 4 prior to machine 6. Trial schedules must satisfy all constraints prior to being evaluated and possibly accepted. The output of the system is a schedule that is acceptable to management and that is on-time for implementation on the shop floor.

The permutation mechanism randomly selects two assignments of jobs to machines and swaps their sequence in the schedule. The most difficult part of the formulation is the calculation of the objective function value. Here, as before, the objective is to minimize the makespan or total production time. The calculation of the production time keeps separate totals for each job and each machine. As new operations are scheduled the totals for both that job and machine get updated to their previous totals plus the processing time for that operation. A check is made for the maximum total (job total or machine total) and both of the

new totals are updated to this maximum value. This is shown below.

$$TJ(P) = TJ(P) + P(L,P)$$

$$TM(L) = TM(L) + P(L,P)$$

$$\text{IF } TJ(P) > TM(L) \text{ THEN } TM(L) = TJ(P), PT = TJ(P) \text{ ELSE}$$

$$TJ(P) = TM(L), PT = TM(L)$$

Here, $TJ(P)$ is the job total array, $TM(L)$ is the machine total array, and PT is the total processing time. Each operation (assignment of a job to a machine) is given a sequence number. Operations are then scheduled from one to N , where N equals the number of jobs multiplied by the number of machines.

Table 6.1 shows results for the job shop scheduling formulation presented in Chapter V. The results show the amount of schedule improvement achievable, indicate the computational times needed, and compare the different versions of the algorithm. All results are from programs run on a machine with a 80386 based microprocessor operating at 22 megahertz.

B. Job Shop Scheduling Results

Table 6.1

Job Shop Results

4 Jobs, 4 Machines (1-20 sec)

<u>Initial Make-span</u>	<u>Final Make-span</u>
46	32
47	25
58	30
39	27
45	27

5 Jobs, 5 Machines (1-1.5 min)

<u>Initial Make-span</u>	<u>Final Make-span</u>
69	40
65	36
63	41
46	33
56	34

6 Jobs, 6 Machines (2-4 min)

<u>Initial Make-span</u>	<u>Final Make-span</u>
73	49
68	41
86	49
74	38
81	44

8 Jobs, 8 Machines (2-8 min)

<u>Initial Make-span</u>	<u>Final Make-span</u>
108	72
105	71
111	77
89	53
110	79

10 Jobs, 10 Machines (5-15 min)

<u>Initial Make-span</u>	<u>Final Make-span</u>
133	102
140	92
148	106
109	84
148	110

12 Jobs, 12 Machines (15-30 min)

<u>Initial Make-span</u>	<u>Final Make-span</u>
175	126
177	133
164	131
159	108
166	121

The initial schedule is generated by an heuristic that provides a fair starting point to attempt to optimize from. Simulated annealing then performs a biased random walk in an attempt to get better schedules. Results indicate that even when we start with a fairly good schedule we can find improvements in a reasonable amount of time.

Figure 6.2 shows a bar chart of the 10x10 instance. This represents the typical improvements achievable from good starting solutions.

JOB SHOP RESULTS

10 JOBS, 10 MACHINES

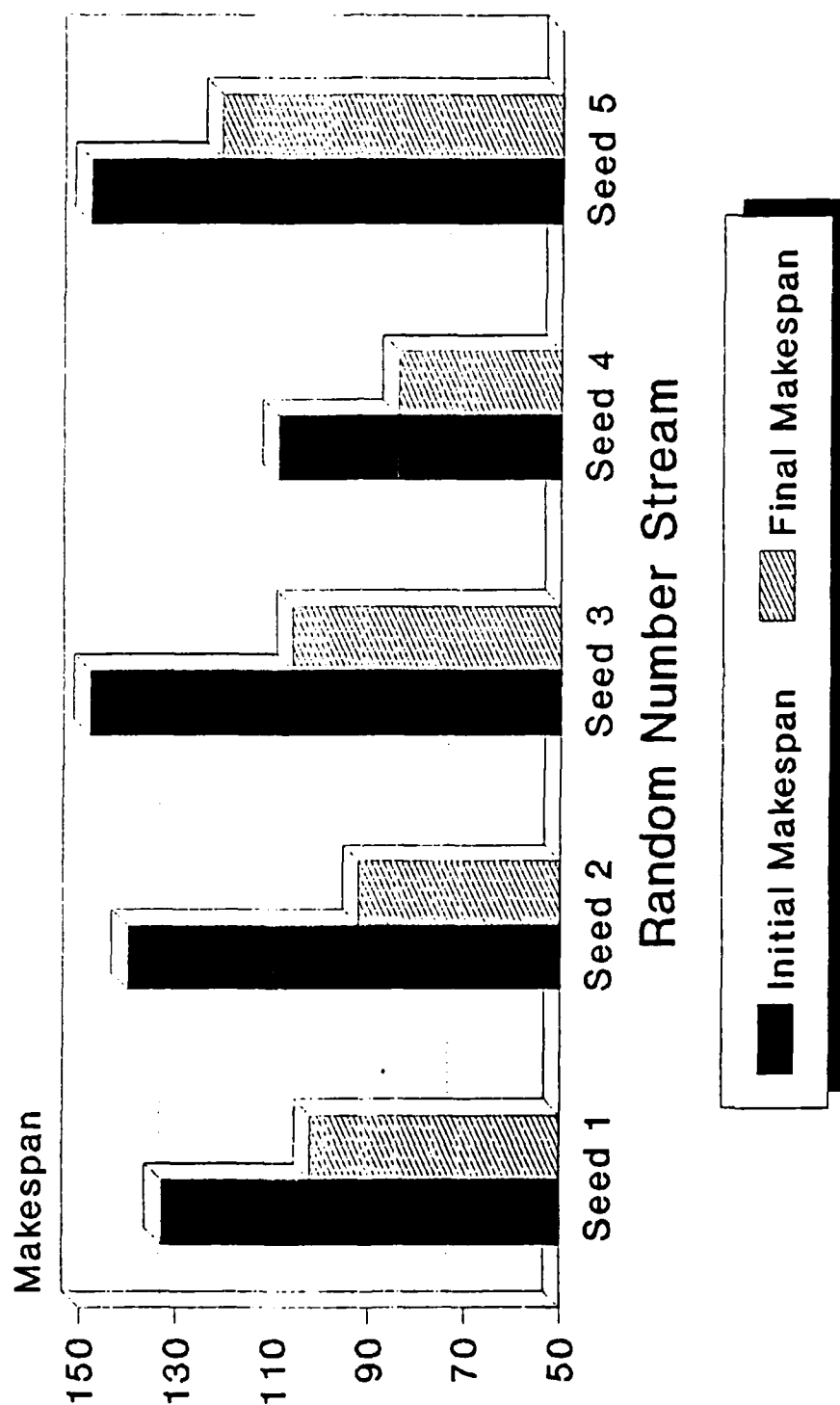


Figure 6.2 Job shop results.

Just as in the flow shop case we test the alternate acceptance functions for the job shop formulation. Here JSU \approx SA-UN, JS1 \approx FS1, and JS2 \approx FS2. Table 6.2 shows the results.

Table 6.2

Job Shop Results for Aternate Acceptance Functions4 Jobs, 4 Machines

<u>JSNE/time</u>	<u>JSU/time</u>	<u>JS1/time</u>	<u>JS2/time</u>
32 / 0:06	32 / 0:03	32 / 0:19	32 / 0:01
25 / 0:12	25 / 0:23	26 / 0:34	25 / 0:22
30 / 0:50	30 / 0:17	31 / 0:21	30 / 0:28
27 / 0:01	27 / 0:01	28 / 0:13	27 / 0:02
27 / 0:17	27 / 0:11	28 / 0:13	27 / 0:04

5 Jobs, 5 Machines

<u>JSNE/time</u>	<u>JSU/time</u>	<u>JS1/time</u>	<u>JS2/time</u>
40 / 1:25	40 / 0:41	44 / 0:51	40 / 0:20
36 / 0:57	36 / 0:37	39 / 0:42	36 / 0:44
41 / 0:53	41 / 0:40	42 / 0:18	42 / 0:45
33 / 0:19	33 / 0:10	33 / 0:09	33 / 0:03
34 / 1:38	34 / 0:32	37 / 1:07	33 / 0:35

6 Jobs, 6 Machines

<u>JSNE/time</u>	<u>JSU/time</u>	<u>JS1/time</u>	<u>JS2/time</u>
49 / 1:36	45 / 1:50	48 / 2:12	45 / 1:15
41 / 1:57	42 / 2:19	46 / 1:12	41 / 1:49
49 / 2:55	45 / 2:04	49 / 2:08	45 / 3:27
38 / 2:39	38 / 2:38	41 / 1:36	38 / 1:51
44 / 4:13	44 / 2:25	49 / 2:19	44 / 1:06

The interesting results are that JS2 performs very well. It not only has very competitive final solutions, but often gets good solutions very quickly. Specifically for the 6x6 instances. Overall results are shown graphically in Figure 6.3.

JOB SHOP RESULTS (FOR ALTERNATE ACCEPTANCE FUNCTIONS)

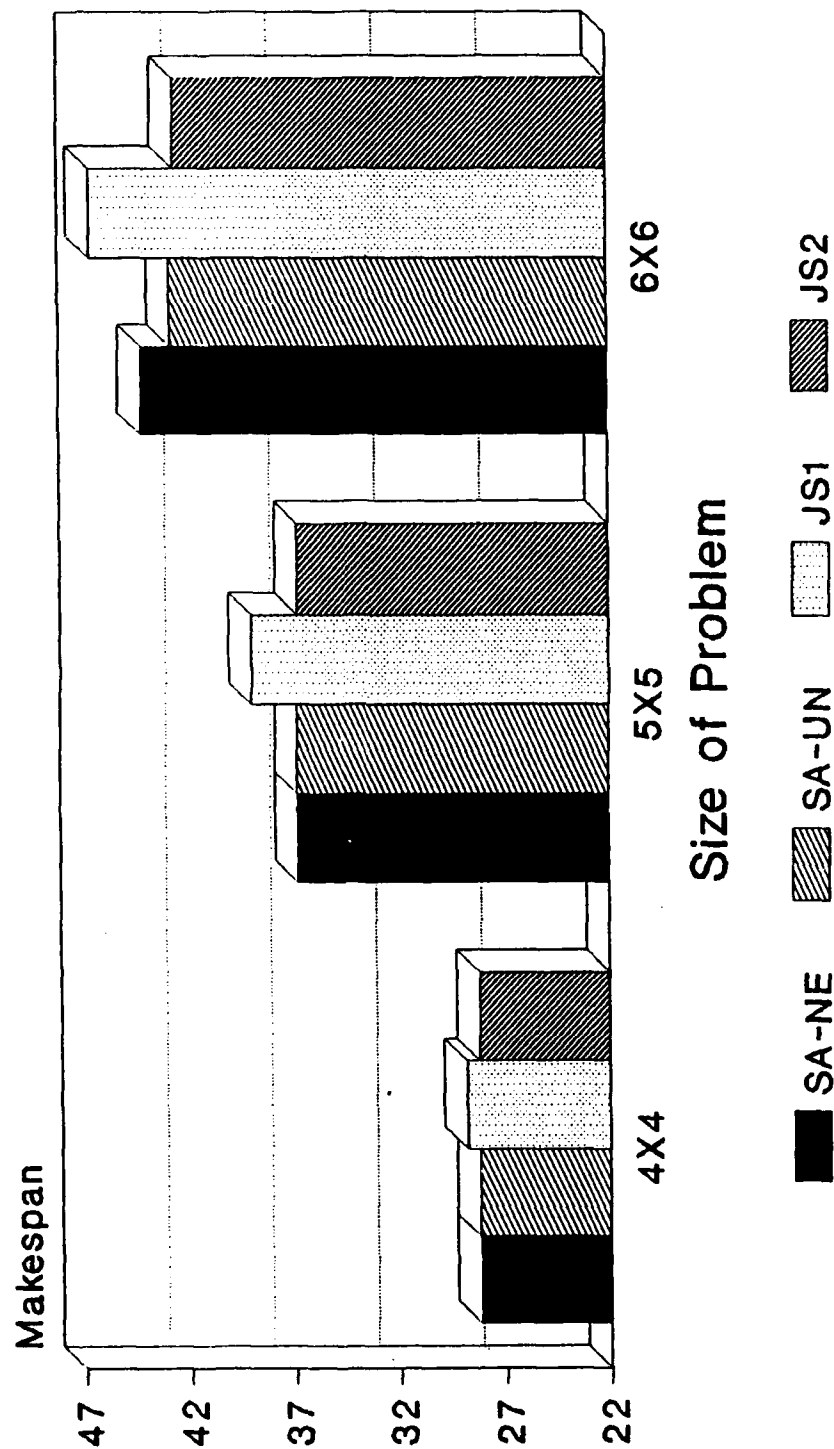


Figure 6.3 Job shop results for alternate acceptance functions.

VII. APPLICATION OF THE MODIFIED SIMULATED ANNEALING ALGORITHM TO THE FLOW SHOP SCHEDULING PROBLEM

A. Flow shop Formulation

In a flow-shop problem all jobs are processed on the various machines in the same order. If job one must be processed on machine 10 prior to machine 12 then the same is true for job five. Similarly, if machine 10 processes job one before job five then the same is true for all machines. The objective is to order the machines in such a way that the completion of the last job is minimized. We want to minimize the makespan.

For an n job m machine problem, let $P=P_1, P_2, \dots, P_n$ be a sequence of jobs and $C_k(P_q)$ be the completion time of job P_q on machine M_k . From the above we have the following relations (Bellman, 1982).

$$C_1(P_q) = C_1(P_{q-1}) + t_{P_q, 1}$$

$$(1) \quad C_k(P_q) = \max_{q-1, 2, \dots, n} \{C_{k-1}(P_q), C_k(P_{q-1})\} + t_{P_q, k},$$

$$q=1, 2, \dots, n; \quad k=2, 3, \dots, m$$

where $C_k(P_0)=0$, $k=1, 2, \dots, m$, and $t_{P_q, k}$ represents the processing time of job P_q on machine k . The only tricky part of the formulation is (1) above. The completion time of job P_q on machine k is the sum of the time to process job P_q on machine k plus the larger of the completion time

of: 1) job P_q on the prior machine $(k-1)$ or 2) the completion time of the previous job (P_{q-1}) on machine k .

We are attempting to minimize $E(P) = C_n(P_n)$. To generate a permutation of the current sequence we simply choose two jobs at random and interchange (swap) their positions in the sequence.

This formulation follows the same modified simulated annealing structure established for the job shop problem. Actual code is found in Attachment E.

B. Simulated Annealing Versus Palmer's Heuristic

Analytical results are from models based on the flow shop formulation presented in Chapter IV. The processing times are randomly generated with different starting seeds. All times are integer values ranging from 1 to 10 units. Table 7.1 shows results for four different methods using various size problems. The approaches employing simulated annealing (SA-NE and SA-UN) have parameter settings: $\alpha=.98$, $T_0=3$, and the temperature is updated for every accepted transition (one repetition at each temperature setting).

Table 7.1

Flow Shop Results Using Simulated Annealing

10 Jobs, 3 Machines			
SA-NE	SA-UN	Palmer's	Local Search
74	74	73*	74
66	66	76	70
60	60	72	60
88	88	92	89
79*	80	90	79
75	75	79	77
10 Jobs, 4 Machines			
66	65*	73	71
74	73*	76	76
67	66*	72	69
89	85*	92	86
83	83	90	83
77	77	79	79
10 Jobs, 5 Machines			
74	74	78	75
78	74*	78	75
75	75	89	77
87	87	91	90
88	88	92	88
87*	88	94	90
10 Jobs, 6 Machines			
80	80	80	80
76	76	82	82
81	81	81	84
98	94*	98	103
100	100	106	103
94*	98	97	95

* indicates a unique "best" solution found

All 24 cases limit the search to two minutes. Column titles are: SA-NE for standard simulated annealing (with a negative exponential acceptance function), SA-UN for

simulated annealing with a uniform acceptance function. The different calculations for the probability of accepting worse configurations (X) are shown below.

$$\text{SA-NE: } X = \exp(D-TD/\text{Temp})$$

$$\text{SA-UN: } X = 1-(TD-D/\text{Temp}),$$

where D is the current configuration's processing duration and TD is that of the trial configuration's. Palmer's method is a widely accepted heuristic (Germain & Sriskandarajan, 1985) that assigns a slope index to the jobs. Slope indexes are highest for jobs that have the strongest tendency to progress from short to long processing times as they pass from machine to machine (French, 1982). Jobs are then ordered in decreasing value of their slope index. Finally, a local search is performed that uses the same permutation scheme and starting order as the simulated annealing approaches, but only accepts transitions that improve the objective function value.

In one of the 24 cases, Palmer's heuristic found a unique best solution. In the remaining 23 cases one of the forms of simulated annealing found the best solution. In six cases the approach using a uniform acceptance function found the unique best while the negative exponential form found it three times. Only two of the cases show the local

search to find better solutions than the standard form of simulated annealing.

The ten job, four machine instance is displayed graphically in Figure 7.1. Here, LS represents results for the local search routine. In this particular instance the pseudo uniform (SA-UN) version is superior to the other three methods.

FLOW SHOP RESULTS

10 JOBS, 4 MACHINES

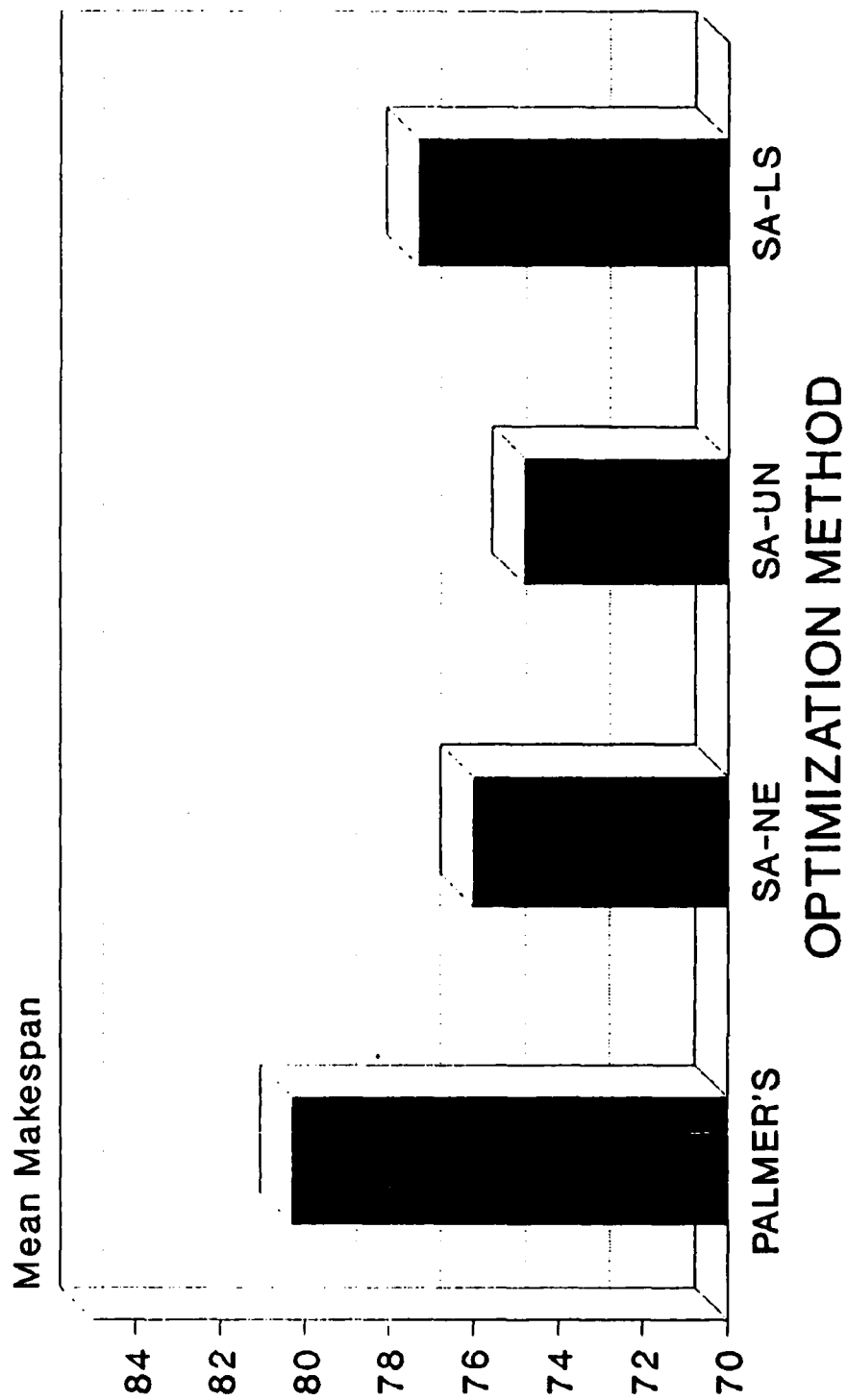


Figure 7.1 Flow shop results.

Using the "hot" key to interrupt the algorithm and monitor its performance shows that the local search routine quickly converges to its local minimum and remains there. The performance of SA-NE and SA-UN are very similar, finding like solutions after the same amounts of processing time.

To investigate the sensitivity of the final solution to the temperature decrement parameter, α , some additional runs are presented. Table 7.2 compares results for settings of .98 and .995.

Table 7.2

Alpha Sensitivity for Flow Shop Problem

10 Jobs , and 5 Machines

SA-UN $\alpha=.98$	SA-UN $\alpha=.995$
74	73
74	78
75	81
87	89
88	88
88	89

10 Jobs , and 5 Machines

80	80
76	85
81	87
94	101
100	103
98	96

Only one of the cases result in a better solution with the alpha set high (slower decrease in the temperature

parameter). On the average, the higher setting yields solutions that are .036 percent below those found using the lower alpha. This is not surprising since the search time is limited to two minutes. At $\alpha=.995$ the algorithm accepts too many inferior configurations and spends too much time bouncing around poor quality solutions.

Table 7.3 shows results for larger problems.

Table 7.3

Flow Shop Results for Large Problems

20 Jobs , and 8 Machines (5 min)

<u>SA-EX</u>	<u>SA-UN</u>	<u>Palmer's</u>
179	180	182

30 Jobs , and 8 Machines (5 min)

<u>SA-EX</u>	<u>SA-UN</u>	<u>Palmer's</u>
241	238	252

40 Jobs , and 8 Machines (10 min)

<u>SA-EX</u>	<u>SA-UN</u>	<u>Palmer's</u>
312	305	314

60 Jobs , and 8 Machines (60 min)

<u>SA-EX</u>	<u>SA-UN</u>	<u>Palmer's</u>
411	407	426

All cases show better solutions found with simulated annealing than Palmer's heuristic. For the first case, the standard form found the best solution. In three of the four cases, the form using an inverted uniform acceptance function found superior solutions.

C. Simulated Annealing Versus Johnson's Algorithm

All the solutions above are for heuristic approaches with no guarantee of optimality. Table 7.4 compares the two forms of simulated annealing versus an optimal solution. All cases are for the two machine case, with Johnson's algorithm providing the optimal solution.

Table 7.4

Flow Shop Evaluation for Known Optimal Solutions

10 Jobs, and 2 Machines (30 sec search)				
Johnson's	SA-NE	% above Opt	SA-UN	% above Opt
55	55	0	55	0
62	62	0	62	0
54	54	0	55*	1.8
85	85	0	87*	2.3
70	70	0	71*	1.4
64	64	0	64	0
20 Jobs, and 2 Machines (1 min search)				
104	104	0	104	0
103	107	3.9	107	3.9
100	101	1.0	102	2.0
147	149	1.4	149	1.4
131	132	0.7	132	0.7
121	121	0	121	0

30 Jobs, and 2 Machines (5 min search)

148	148	0	148	0
139	154	10.8	154	10.8
168	168	0	168	0
196	196	0	198	0.10
190	191	0.5	191	0.5
191	191	0	191	0

40 Jobs, and 2 Machines (5 min search)

202	208	2.9	206	2.0
-----	-----	-----	-----	-----

50 Jobs, and 2 Machines (5 min search)

254	254	0	256	0.8
-----	-----	---	-----	-----

60 Jobs, and 2 Machines (5 min search)

308	310	0.6	309	0.3
-----	-----	-----	-----	-----

With only ten jobs the processing time is limited to 30 seconds and α is set to .98. With more jobs, thus more configurations possible, α is increased (.99 for 20 jobs and .995 for 30 jobs) and the search time is increased. Of the 21 cases the optimal solution is found 14 times. Only one case results in a solution more than 4 percent above the optimal. Even in the large cases (40, 50, and 60 jobs) solutions are within 3 percent of the optimal. With 50 jobs the optimal is found in three minutes.

Figure 7.2 shows various instances of this problem using the first random number stream. As the instance size grows, there is a greater likelihood that the algorithm gets trapped in a suboptimal local minima.

FLOW SHOP RESULTS

SA vs JOHNSON'S
(XT Computer)

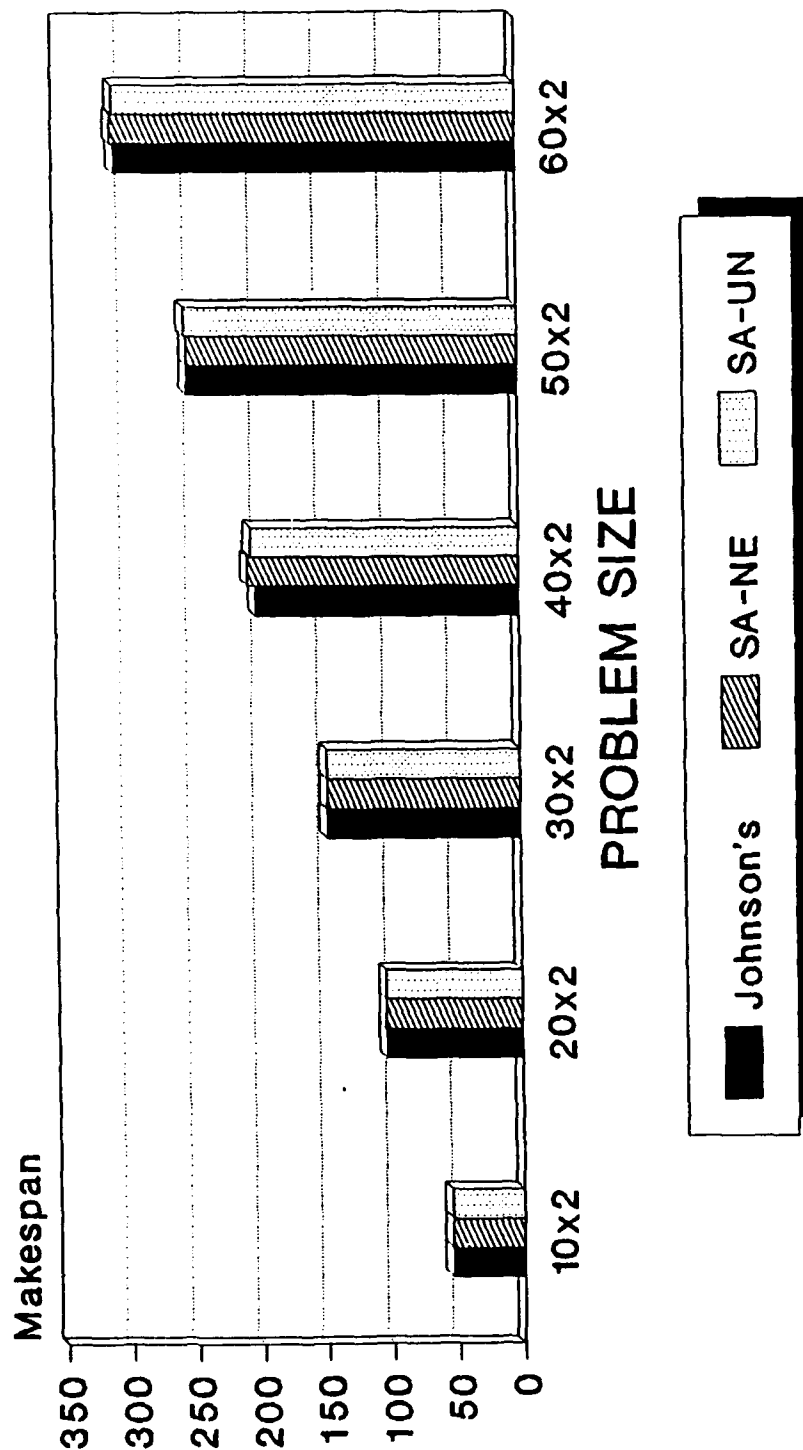


Figure 7.2 Flow shop results (XT).

We must examine the probability of acceptance when attempting to explain the differences in performance for the two versions of simulated annealing. Since their starting configurations and permutations are identical, the only difference is their acceptance function.

Table 7.5

Flow Shop Results for Alternate Acceptance Functions

10 Jobs, 3 Machines			
NE/time	UN/time	FS1/time	FS2/time
74 / 0:29	74 / 0:30	74 / 0:27	74 / 0:18
66 / 1:01	66 / 0:17	67 / 0:21	67 / 1:10
60 / 0:38	60 / 0:46	65 / 0:25	60 / 0:24
88 / 0:57	88 / 0:32	88 / 0:05	88 / 0:05
79 / 1:16	80 / 0:01	79 / 0:14	79 / 0:03
75 / 0:19	75 / 0:43	76 / 0:14	76 / 0:05

10 Jobs, 4 Machines			
NE/time	UN/time	FS1/time	FS2/time
66 / 1:30	66 / 1:12	69 / 0:33	69 / 2:07
74 / 1:25	74 / 0:46	72 / 1:48	73 / 1:19
67 / 2:00	67 / 0:35	67 / 1:14	68 / 1:06
89 / 1:49	89 / 0:33	90 / 0:51	92 / 0:19
83 / 0:42	83 / 1:08	83 / 0:58	83 / 0:40
77 / 1:00	77 / 0:40	77 / 1:18	79 / 1:18

10 Jobs, 5 Machines

<u>NE/time</u>	<u>UN/time</u>	<u>FS1/time</u>	<u>FS2/time</u>
74 / 0:43	74 / 1:47	76 / 0:09	75 / 2:06
78 / 0:42	74 / 1:08	73 / 2:37	77 / 1:38
75 / 0:54	75 / 0:38	75 / 1:32	80 / 1:02
87 / 1:10	87 / 0:53	85 / 2:14	90 / 1:20
88 / 1:37	88 / 0:49	88 / 1:06	88 / 1:32
87 / 1:12	86 / 2:14	86 / 2:26	92 / 1:56

Although the algorithms appear to perform similarly, as the problem size grows, the standard form has greater opportunity to find quality solutions and the other forms tend to get stuck more. The other versions do find fairly good solutions quickly.

D. Large Flow Shop Problems

To investigate the algorithm's performance for larger problems the code is converted to FORTRAN and executed on a CRAY X-MP/14se computer. Table 7.6 shows results compared to Palmer's heuristic and gives the running times for the standard version of simulated annealing.

Table 7.6

Flow Shop Results from the CRAY Computer20 Jobs, 20 Machines

<u>Makeshift</u>		<u>Run Time(sec)</u>	<u>Palmer's</u>
<u>Initial</u>	<u>Final</u>		
306	242	2.18	276
316	254	.35	272
295	252	.34	281
294	247	.47	267
280	243	.82	282

30 Jobs, 30 Machines

<u>Makeshift</u>		<u>Run Time(sec)</u>	<u>Palmer's</u>
<u>Initial</u>	<u>Final</u>		
439	382	1.27	432
454	389	1.03	427
439	376	1.52	425
428	365	1.62	400
464	381	1.59	417

40 Jobs, 40 Machines

<u>Makeshift</u>		<u>Run Time(sec)</u>	<u>Palmer's</u>
<u>Initial</u>	<u>Final</u>		
619	531	2.36	577
616	530	2.59	581
616	520	2.65	565
605	508	3.06	565
604	520	2.79	581

The results show that in all of the 15 cases examined the simulated annealing algorithm finds superior solutions than Palmer's heuristic. CPU time is also minimal with the worst case being 3.06 seconds.

In the 40x40 instance, average solution improvements of 15 percent are achieved in approximately 2.7 seconds. This assumes starting from a random configuration. When compared to Palmer's heuristic, simulated annealing finds solutions that average ten percent better. These results are presented graphically in Figure 7.3.

FLOW SHOP RESULTS

40 JOBS, 40 MACHINES
(CRAY COMPUTER)

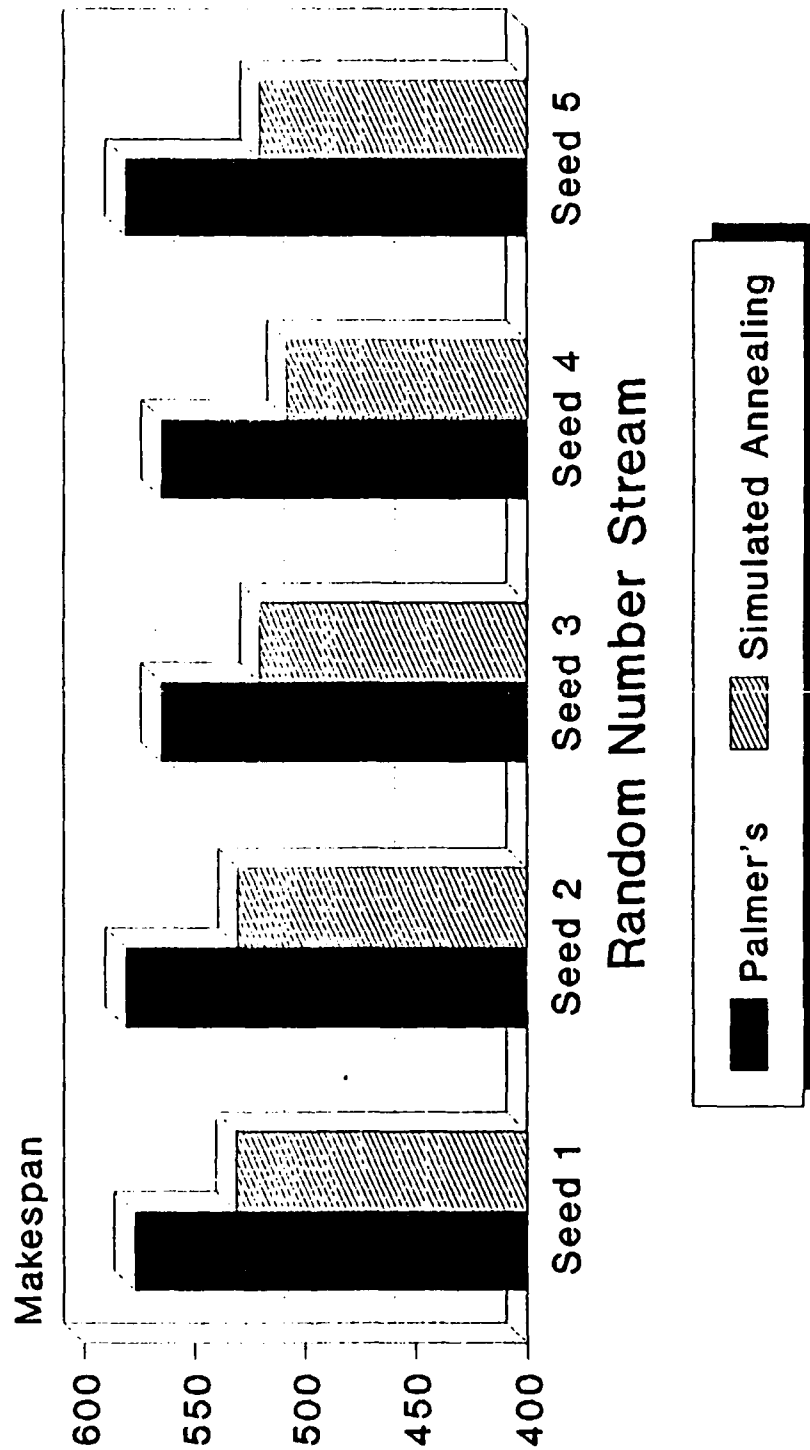


Figure 7.3 Flow shop results, 40x40 problem.

Table 7.7 provides results for large instances of the flow shop problem where known optimal solutions are available. As before, the programs are in FORTRAN and executed on a CRAY computer. There are four cases, employing different random number streams. The three optimization techniques are Johnson's algorithm, simulated annealing with a negative exponential acceptance function, and simulated annealing with a pseudo uniform acceptance function.

Table 7.7

Flow Shop Results, Large Instances (on a CRAY)100 Jobs, 2 Machines

<u>Johnson's</u>	<u>SA-NE</u>	<u>Time</u>	<u>SA-UN</u>	<u>Time</u>
531	572	.046	572	.061
568	568	.050	568	.050
575	575	.069	575	.105
569	587	.041	587	.045

400 Jobs, 2 Machines

<u>Johnson's</u>	<u>SA-NE</u>	<u>Time</u>	<u>SA-UN</u>	<u>Time</u>
2245	2245	.272	2245	.192
2245	2271	.140	2270	.212
2092	2181	.187	2181	.142
2162	2200	.167	2200	.145

500 Jobs, 2 Machines

<u>Johnson's</u>	<u>SA-NE</u>	<u>Time</u>	<u>SA-UN</u>	<u>Time</u>
2857	2857	.271	2857	.250
2836	2836	.224	2837	.176
2779	2779	.226	2779	.157
2703	2713	.208	2713	.157

Results are presented graphically, as well, in Figure 7.4.

Notice that better solutions are found with the larger instances of 400 and 500 jobs. This shows the importance of trial and error runs to find appropriate parameter settings needed to prevent the algorithm from getting trapped in a suboptimal local minima.

FLOW SHOP RESULTS

500 JOBS, 2 MACHINES
(CRAY COMPUTER)

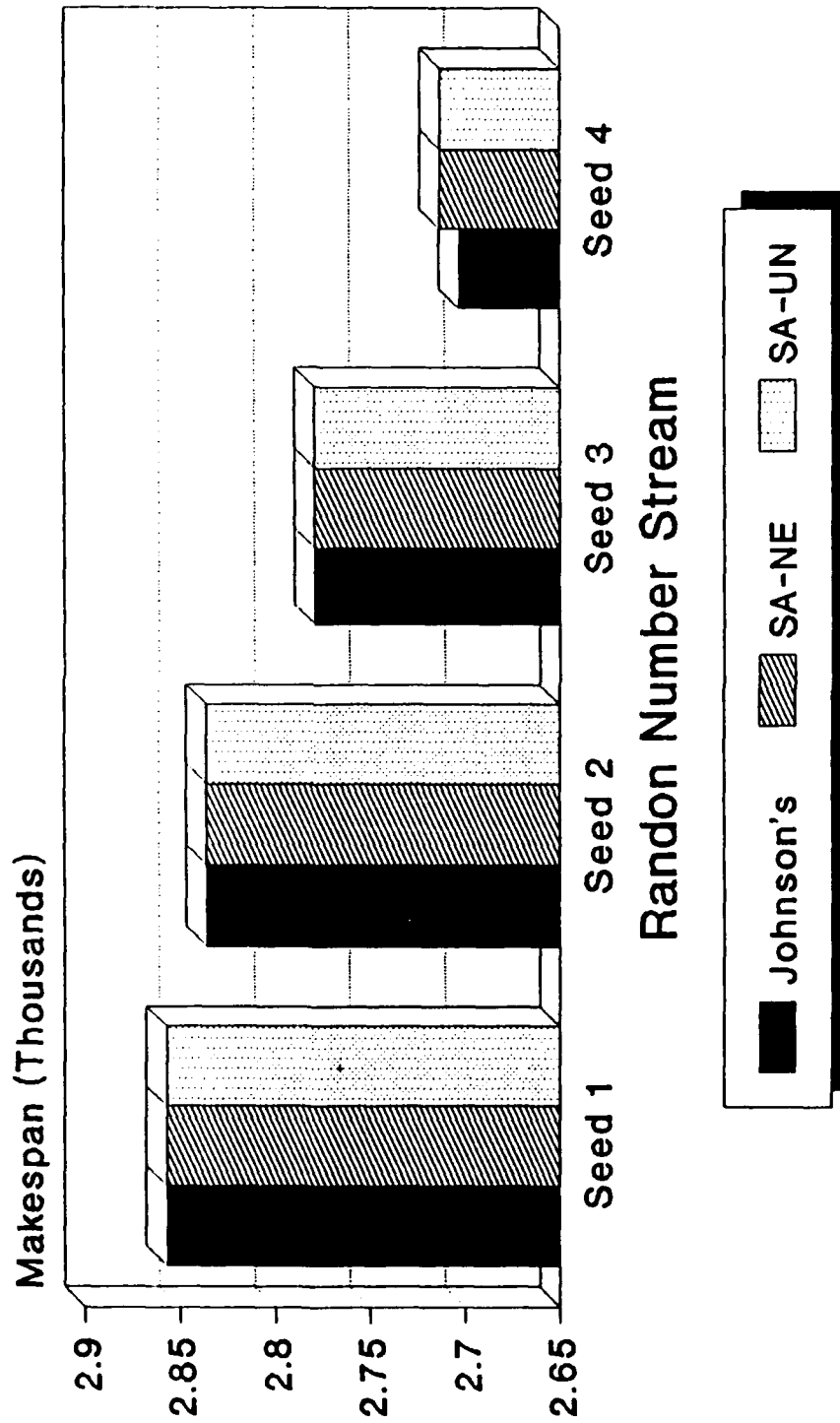


Figure 7.4 Flow shop results (on CRAY).

VIII. SIMULATED ANNEALING'S POTENTIAL FOR OTHER INDUSTRIAL ENGINEERING PROBLEMS

A. Types of Problems Appropriate for Simulated Annealing

Aarts and Van Laarhoven (1989) have grouped a number of problems solved by simulated annealing into one of 13 types. These categories are below:

1. Traveling Salesman Problems
2. Matching Problems
3. Graph Partitioning Problems
4. Quadratic Assignment Problems
5. Linear Arrangement Problems
6. Graph Colouring Problems
7. Scheduling Problems
8. VLSI Design Problems
9. Facilities Layout
10. Image Processing
11. Code Design
12. Biology
13. Physics

The authors further state that simulated annealing has been applied to optimization problems with continuous variables.

As previously stated, one of the measures of a good heuristic is its flexibility in handling variations of a problem. As indicated by the above list simulated annealing

gives its user the ability to address a broad range of problem types. Another measure of a heuristic is its ease of implementation. Although the algorithm itself is rather simple to implement, the process of reformulating the problem into an equivalent format prior to using simulated annealing is not always trivial.

B. Example Formulations

This section brings together the various components for selected problems. The TSP is presented in detail, followed by a less thorough treatment of two additional formulations. Appendices A-G contain BASIC code for the respective problems, while H shows three additional formulations.

1. The Traveling Salesman Problem Formulation

Aarts and Korst (1989) provide a concise description of the general problem formulation. He defines the TSP problem as follows. Let n be the number of cities and $D=[d_{ij}]$ be the distance matrix whose elements d_{ij} denote the distance between cities i and j . The Problem is to find the shortest route visiting each city exactly one time. The solution space S is the set of all cyclic permutations $\pi=(\pi(1),\dots,\pi(n))$, where $\pi(i)$, $i=1,2,\dots,n$, denotes the successor city of city i in the route represented by π . The cost function to be minimized is:

$$c(\pi) = \sum_{i=1}^n d_{i, \pi(i)}.$$

Permutations can be generated by randomly selecting two cities, p and q , and reversing the direction between them. The difference in the cost for such a permutation is calculated from:

$$\delta c = -d_{p, \pi(p)} - d_{\pi^{-1}(q), q} + d_{p, \pi^{-1}(q)} + d_{\pi(p), q}.$$

Figure 8.1 shows the pseudo code, based on the BASIC code found in attachment A.

BEGIN

INITIALIZE

Set Temp (T_0)

Generate Route ($R_0(N)$)

Calculate Distance (D_0)

Set Temp Decrement (α) and Repetitions Increment (β)

Set Number of Trials Counter (NT) 'stop criterion

REPEAT

For $L=1$ to L_k

Begin ' Permute Temporary Route ($TR(N)$)

Generate 2 Random Integers(1-N), J and K

Reverse Direction Between cities J and K

Calculate Distance for Temporary Route (TD)

If $TD \leq D$ Then UPDATE Else

If $\exp(D-TD/T) > \text{Random}(0,1)$ Then UPDATE

End

$L_k = L_k * \beta$

$T = T * \alpha$

UNTIL Stopcriterion (NT) is met

END

UPDATE

Reset Current Route and Distance

$$R(N) = TR(N)$$
$$D = TD$$

RETURN

Figure 8.1 Pseudo code for simulated annealing algorithm applied to the TSP.

We begin by initializing the various parameters and generating the initial route. In attachment A, the code used for this formulation generates an initial route by simply placing city one in position one, city two in position two, etcetera along the route. The purpose of β is to increase the number of repetitions, or accepted permutations, at each control parameter setting. The number of trials (NT) parameter is used as a check to determine if the algorithm has failed to find an acceptable permutation after NT attempts. If no acceptable permutation is found, then we are reasonably close to the optimal and the algorithm terminates. Permutations are generated by the reversal method discussed previously. Two random positions are selected along the route and the direction between the cities occupying those positions is reversed.

2. The Assignment Problem

Assignment problems are very similar to the TSP, but much less restrictive. Here, subtours are allowed. For instance, the optimal assignment matrix below is acceptable.

		MACHINE			
		1	2	3	4
M A N	1	[X		
	2				
	3				X
	4			X	
]			

Clearly, this does not represent an acceptable solution to a TSP. For this problem the objective function is:

$$\sum_{i=1}^n \sum_{j=1}^n C_{ij} X_{ij}$$

Subject to the constraints:

$$\sum_{j=1}^n X_{ij} = 1, \quad i=1, \dots, n.$$

$$\sum_{i=1}^n X_{ij} = 1, \quad j=1, \dots, n \text{ (French 1985).}$$

Figure 8.2 illustrates the use of simulated annealing for the assignment problem. The permutation is performed by making a one-for-one swap of two randomly generated assignments. Pseudo code for an N by N matrix is shown below:

INITIALIZE:

generate initial set of assignments

for I=1 to N

for J=1 to N

select the lowest cost element in row I

remove col J and record its position

next J

next I

from distance matrix calculate total distance, D

GEN: if timer > TIME then end

generate J and K, two random integers from 1 to N

swap column positions for rows J and K

calculate the difference in distances, DL

if $DL \leq 0$ then update

$x = \exp(-DL/\theta)$

if $x > \text{rnd}(0,1)$ then update

swap back column positions for rows J and K

goto gen

UPDATE: $D = D + DL$

$\theta = \theta * \alpha$

goto gen

```
END:  print assignments and D  
      stop
```

Figure 8.2 Pseudo code for an assignment problem.

This simulated annealing formulation is very similar to the TSP formulation, with the exception of the permutation to the current configuration. The assignment problem employs the swap permutation. Two random assignees are selected and their corresponding assignments are exchanged. Attachment B contains the code for this formulation.

3. Assignment Problem Results

Thirty cases of the assignment problem are evaluated. Each case generates a square matrix of assignment costs, which are random integers from one to ten. The permutation mechanism simply swaps two randomly selected assignments. Smaller problems (less than 20) are solved on an IBM XT using a 8088 processor and larger problems are solved on a 80386 processor. Table 8.1 provides results.

Table 8.1

Results of the Assignment Problem10 Assignments (XT)

<u>Simulated Annealing</u> / <u>Time</u>	<u>Hungarian</u>
21 :54	21
16 1:41	16
21 2:25	21
11 1:40	11
29 1:20	29

12 Assignments (XT)

<u>Simulated Annealing</u> / <u>Time</u>	<u>Hungarian</u>
24 2:40	24
17 1:39	17
22 2:44	22
18 2:21	18
26 3:46	26

15 Assignments (XT)

<u>Simulated Annealing</u> / <u>Time</u>	<u>Hungarian</u>
24 5:00	24
20 8:27	20
23 2:51	23
18 5:00	18
28 4:46	28

20 Assignments (386)

<u>Simulated Annealing</u> / <u>Time</u>	<u>Hungarian</u>
27 :14	27
25 :20	25
28 :07	28
25 2:03	25
28 :36	28

30 Assignments (386)

<u>Simulated Annealing</u> / <u>Time</u>	<u>Hungarian</u>
35 5:40	35
32 3:18	32
34 6:45	34
32 6:26	32
33 2:59	33

50 Assignments (386)

<u>Simulated Annealing</u> / <u>Time</u>	<u>Hungarian</u>
50 6:18	50
51 8:05	51
50 5:37	50
51 4:51	51
50 8:10	50

In all 30 cases tested the optimal set of assignments was found. For cases with fewer than 20 assignments an IBM-XT processor found the optimal. Only one of the 15 problems required more than five minutes. For problems with 20 or more assignments an 80386 processor found the optimal set of assignments. In all 15 cases the optimal set is found within approximately eight minutes.

Figure 8.3 shows how the algorithm performs over time. The 50 node assignment problem is displayed, showing cost (objective function value) improvements relative to processing time on a 80386 machine. Notice the large initial improvement and gradual convergence to the optimal solution, typical of the algorithm's performance.

Additional formulations for zero-one programming, continuous functions, and linear programming are found in Appendix H.

ASSIGNMENT PROBLEM RESULTS

50 NODES
(Optimal = 50.4)

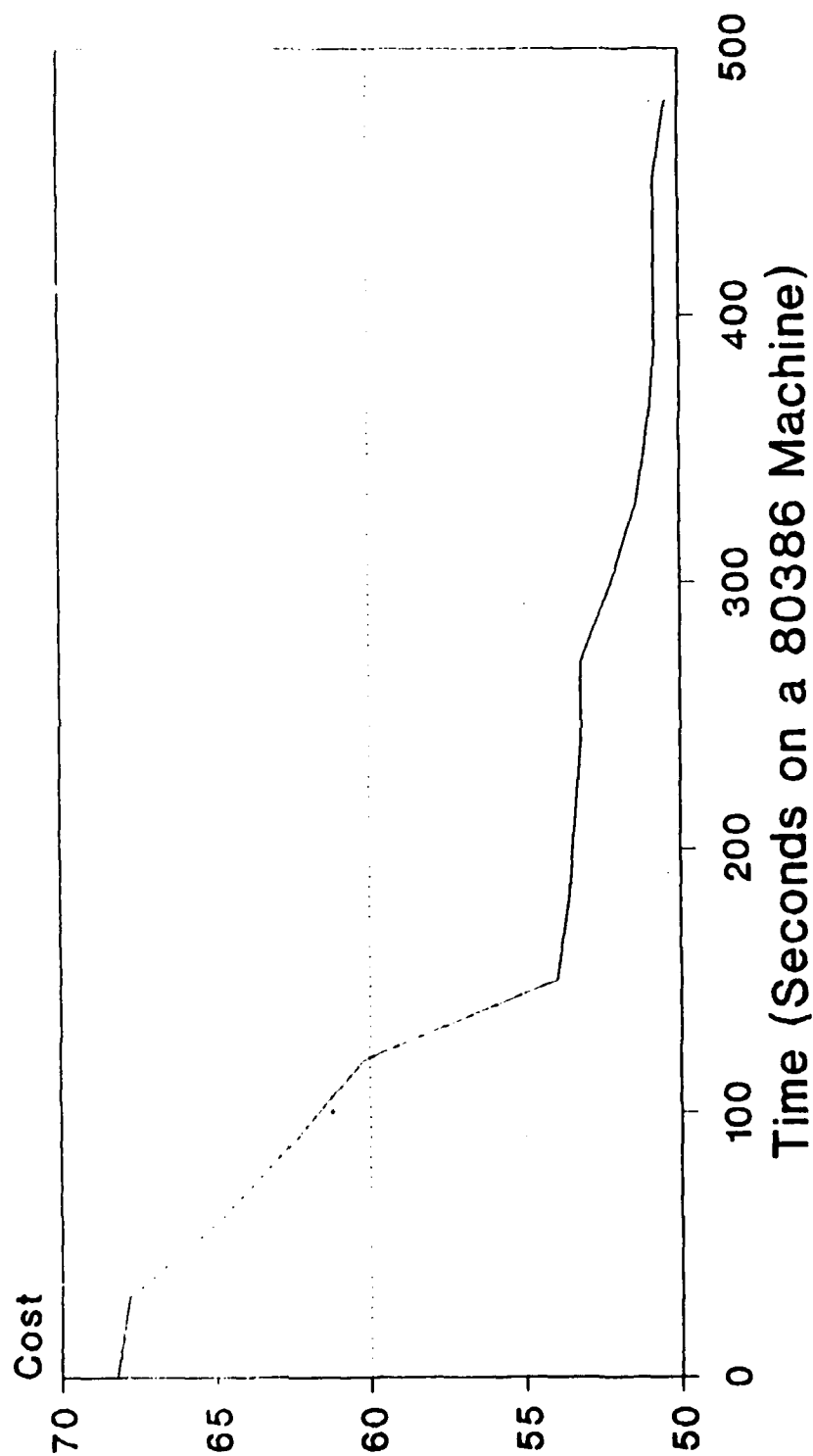


Figure 8.3 Assignment problem results, N=50.

4. Minimum Cut Formulation

Minimum cut and maximum cut problems are subsets of the larger class known as graph partitioning problems. These can be either weighted or non weighted and have great practical relevance (Ullman 1984). Aarts and Korst (1989) provides a general formulation that is modified for the minimum cut problem, shown below.

Given a graph $G = (V, E)$ with positive weights on its edges, find a partition of its vertices (V) into two disjoint sets V_0 and V_1 such that the sum of weights of edges with one vertex in V_0 and the other in V_1 is minimal. Simulated annealing is applied by defining the solution space, the cost function, and a permutation mechanism. The solution space consists of all possible partitions of V into V_0 and V_1 . The cost function, to be minimized, is:

$$f(V_0, V_1) = \sum_{\{u, v\} \in \delta(V_0, V_1)} w(\{u, v\}),$$

where $w(\{u, v\})$ denotes the weight of edge $\{u, v\}$, and $\delta(V_0, V_1)$ is the cut of a partition of V defined by:

$$\delta(V_0, V_1) = \{\{u, v\} \in E \mid u \in V_0 \cap v \in V_1\}.$$

Permutations are generated by randomly selecting one vertex from one of the two sets and placing it in the opposite set.

For example, select $u' \in V_0$ is placed in V_1 . The cost difference is computed below.

$$\delta f = \left(\sum_{(u', v) \in E \setminus \delta(V_0, V_1)} w(\{u', v\}) - \sum_{(u', v) \in \delta(V_0, V_1)} w(\{u', v\}) \right)$$

This notation adds the weights of the newly created edges and subtracts the weights of all edges removed by the new partition. When partitioning the sets the start and end nodes must always remain in sets V_0 and V_1 respectively. The algorithm's flow follows the same pattern of previous formulations. Actual code is found in Attachment D.

C. Faster Implementations

As we have seen, there are a number of advantages associated with the simulated annealing algorithm. It finds near-optimal solutions, it is easy to implement, and it is flexible. The disadvantage is that it takes a great deal of time to asymptotically converge to optimality. We shall discuss three approaches to increasing its speed, with primary emphasis on parallel processing.

1. Faster Sequential Algorithms

Aarts and Korst (1989) identify two areas that offer opportunities for speeding up the basic sequential algorithm. Focus is on the generation mechanism (generates neighboring permutations) and the cooling schedule. Green and Supowit (1986) provide the "generate less" method which

creates shorter length Markov chains for a number of problems. This method depends strongly on the neighboring structure of permutations and is not applicable for combinatorial optimization problems. Cooling schedules can also be tailored for the problem at hand. However, this approach generally does not result in large savings in computational time. Catthor, DeMan, and Vanderwalle (1988) provide an example for problems that exhibit clustering in their solution space.

2. Hardware Accelerators

The idea here is to provide dedicated hardware for the time consuming portions of the algorithm. Iosupovici, King, and Breuer (1983) use dedicated hardware for evaluating the cost of a trial configuration for a wire placement problem. Another scheme for placement problems is offered by Spira and Hage (1985), who report increase in speed ranging up to 20 percent. They rewrite the time consuming portions of the algorithm in micro code and execute it on a fast micro engine attached to the host computer.

3. Parallel Processing

Aarts and Korst (1989) give a good account of designing simulated annealing algorithms for parallel machines. The parallel algorithm must distribute the execution of the various parts over a number of communicating parallel processors. This is not a trivial task due to the sequential nature of the simulated annealing algorithm. Transitions from one configuration to another are attempted serially. According to Flynn (1966), there are four basic types of parallel-machine models. These are shown below:

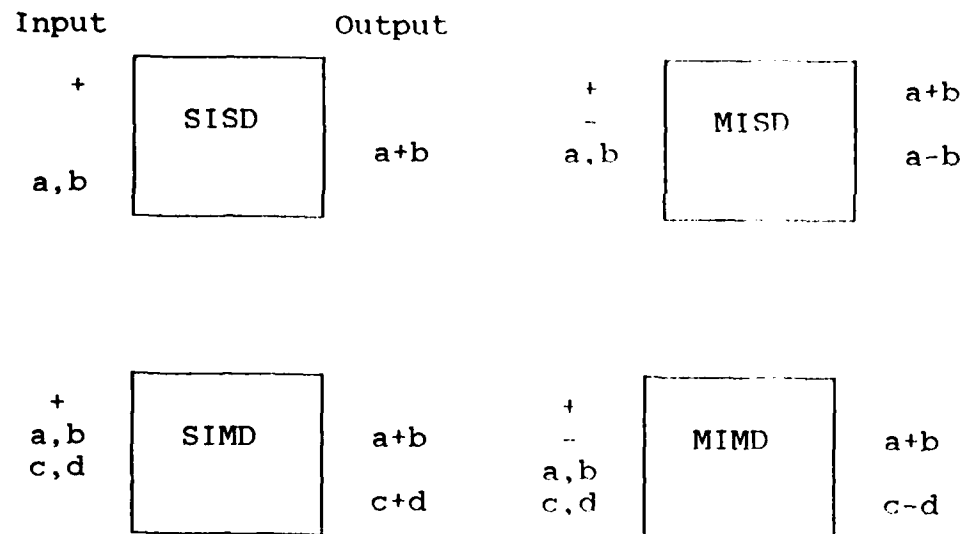


Figure 8.4 A classification of parallel-machine models according to Flynn (1966).

Flynn describes these four general classes as:

- SISD (Single Instruction, Single Data): one instruction at a time is executed on one set of data. This class includes the classical sequential computer.

- SIMD (Single Instruction, Multiple Data): one instruction at a time is executed on multiple sets of data. This class includes vector computers and array processors.

- MISD (Multiple Instructions, Single Data): multiple instructions at a time are executed on one set of data. This class has received little attention.

- MIMD (Multiple Instructions, Multiple Data):
multiple instructions at a time are executed on multiple sets of data. The processors in a MIMD machine are either synchronized, performing each successive set of instructions simultaneously, or unsynchronized, performing all instructions independently. This class is currently receiving a great deal of attention.

In converting sequential simulated annealing algorithms to parallel algorithms the focus is on the generation mechanism. Aarts and Korst (1989) state that the evaluation of a trial configuration consists of the following four tasks.

1. Selecting a new configuration from the neighbors of the current configuration.
2. Calculating the cost difference between the solutions to the two configurations.
3. Accepting or rejecting the new configuration.
4. Replacing the new configuration and solution if step three results in an acceptance.

Examining these four steps in detail leads Aarts to two observations.

1. The first three steps can be performed in parallel for different trials, since they are essentially independent. Executing the fourth step in parallel creates several problems. It is impossible to replace accepted solutions in parallel. Say processors A and B both accept transitions from the current configuration. Transitioning to configuration A is performed first. Now, the transition to configuration B will be made from the new configuration A which results in an unknown configuration and a potentially poor solution. As a result, correct local decisions may lead to incorrect global decisions.

2. Based on the typical performance of the simulated annealing algorithm, we know that the ratio between the number of executions of steps 1-3 and the number of times step 4 is executed will change during the running of the algorithm. This ratio is called the acceptance ratio. It will be close to one when the control parameter, c , is high and approaches zero as c approaches zero. Thus, the relative frequency of step 4 decreases as the algorithm runs longer. This provides the opportunity to use different parallel algorithms in different regimes of the control parameter.

Two strategies are given by Aarts: single-trial parallelism and multiple-trial parallelism. With single-trial parallelism, the task of evaluating the single trial

is divided over the number of processors. The amount of speed-up achievable is a function of the type of problem. Speed-up is defined by Aarts as the running time of the sequential algorithm executed on one processor divided by the running time of the parallel algorithm executing on several processors. If the generation and evaluation of new configurations can be divided into independent subtasks, then the speed-up is significant. However, for the TSP and many other combinatorial optimization problems, this is not the case (Aarts & Korst, 1989).

Multiple-trial parallelism is a routine that evaluates trial configurations simultaneously. Aarts provides the following scenario. Many processors simultaneously generate trial configurations from the current one. This process is repeated until one of the processors accepts a new configuration. At this point execution is frozen until all processors are reset to the new configuration. Execution resumes with the processors generating trials from the new current configuration. If more than one processor accepts a trial at the same time, then an arbiter decides which one to select. Using this scheme, the speed-up is small at low values of the control parameter, since most trial configurations are accepted and in essence one processor at a time is active. As the value of the control parameter decreases, the speed-up increases and eventually becomes proportional to the number of processors. An example of a

placement and routing problem for VLSI design is found in Darema-Rogers, Kirkpatrick, and Norton (1987) and Banerjee and Jones (1986).

A good example that combines both forms of parallelism is reported by Kravitz and Rutenbar (1987). In the regime where the control parameter is high they use a single-move decomposition algorithm, based on single-trial parallelism. Tasks involved in generating and evaluating a trial configuration are divided into several subtasks that are performed in parallel. In the regime associated with small values of the control parameter they switch to a parallel-move algorithm, a form of multiple-trial parallelism. They report the speed-up of the single move decomposition algorithm is a constant (≈ 2) as a function of the control parameter. The speed-up for the parallel-moves algorithm is small (≈ 1) in the large-value regime and large (≈ 3.5) in the small-value regime of the control parameter. This is for implementation on a four-processor system.

The algorithms presented execute those tasks in parallel that are independent. These are tasks 1-3 identified earlier. These algorithms are suited for execution on synchronous MIMD machines (Aarts & Korst, 1989).

IX. CONCLUSION

A. Summary

In this is paper we added to the body of knowledge a method that takes advantage of a biased random walk. More specifically, we introduced the modified simulated annealing algorithm as a means of biasing a random walk to move in the direction of more favorable solutions to combinatorial optimization problems. To formulate a problem we need a concise description of the configuration of the system, a random permutation mechanism, an objective function value, and an annealing schedule. Modifications to the standard formulation include: a front end expert system, a constraints module, a user interrupt capability, and the use of alternative acceptance functions. The algorithm behaves much like a local search, with one exception, it probabilistically allows transitions to less optimal states. Thus, it allows the algorithm to escape local minimas (minimization problems). Although simulated annealing has its foundations in the field of statistical mechanics, we have seen that the algorithm performs well with alternative acceptance functions. Specifically, for the flow shop and job shop cases, the various acceptance functions performed as well or better than the standard formulation. The seemingly mystical connection between combinatorial optimization problems and statistical mechanics is best summed up by N.S. Krylov (1979) "the problem of establishing

a connection between statistics and mechanics should be regarded as being absolutely unsolved."

A simpler explanation of the algorithm's behavior is that of a biased random walk. The user can provide many elements that introduce bias. Some of these are: the various parameter settings (α , β , T_0), the acceptance function, and constraints regarding acceptable solutions. The families of acceptance functions we introduced in Chapter V represent ways to bias the random walk. Depending on the users' willingness to sacrifice the degree of optimality for the timeliness of solutions, an appropriate acceptance function can be selected. The final solution will differ depending on the collective bias of the user. Kirkpatrick, Gelatt, and Vecchi (1983) conclude that the annealing schedule may be arrived at through trial and error for a given problem. Cerny (1985) states that there is perhaps more poetry than mathematics when selecting the best set of parameters for running the algorithm on a particular problem. In general, problems with many low lying local minimums allow the use of functions with steep (slopes) decreases in their probability of acceptance. The uniform and FS1 functions work well. Alternatively, problems with few local minimums require the use of functions with more gradual decreases in their probability of acceptance. Thus, allowing more search time to be spent near the asymptote. Here, the standard version and JS2 do best.

It is easy to envision an application with a dedicated processor continuously running the simulated annealing algorithm. The algorithm can be interrupted to allow for updated shop floor data. It clearly lends itself to applications in dynamic environments. When a solution is needed, the user simply queries the algorithm to get the best solution found. The algorithm is capable of finding solutions over a variable amount of time. Feasible solutions are available for decisions that are needed quickly. With more decision time, better solutions are achievable. This scenario assumes the delegation of some of the decision making. In a very chaotic scenario, appropriate for the implementation of simulated annealing, Tom Peters (1987) suggests delegation of decision making and sharing of data.

Simulation was introduced in the 1960's as a good analysis tool appropriate when more analytical approaches fail. Similarly, the simulated annealing algorithm is a good optimizer when you can not afford to wait for an optimal solution or when good heuristics do not exist. In a sense we are using the computer's speed to solve problems that to date have not been solvable. The major drawback to the implementation of simulated annealing is that the algorithm takes a long time to converge to optimality. On the positive side, there is a slow increase in effort with an increase in the problem size and the generality of the

algorithm makes it a widely applicable heuristic optimization technique (Kirkpatrick, Gelatt, & Vecchi, 1983).

With the modified formulation presented in this work, the user has the option of selecting acceptance functions that trade computational time for solution quality. Constraints can be input to prevent unacceptable solutions. With limited processing time available, the uniform acceptance function outperforms the standard form of simulated annealing. More specifically, solution quality is as good or better and computational time is less. For the job shop formulation, the acceptance function titled JS2 outperforms both the standard version and the uniform version.

B. Recommendation for Future Work

There is a great deal of work to be done in converting the sequential form of the algorithm to parallel versions capable of running on parallel architecture machines. Cerny (1985) concludes that these algorithms might be even more appropriate for the next generation of multiprocessing machines, where different degrees of freedom can be treated simultaneously. Another approach, discussed in Chapter IV, involves ten processors each generating permutations from the existing configuration. Some arbitrator would break ties and the current configuration gets updated. Initially, at high values of the control parameter, speed-ups would be

slight. As the control parameter is decreased the speed-ups achievable would be much greater. Finally, the greatest speed-ups are achieved where the algorithm takes the longest, during convergence.

The algorithm could be formulated to evaluate PERT/CPM problems. One of the biggest problems experienced by these approaches is the lack of credible input estimates of the task durations. Simulated annealing could accept updated estimates, perform the optimization, and make time and cost tradeoffs.

This paper only looked at three alternative families of acceptance functions. The same approach could be used to evaluate other forms of acceptance functions.

REFERENCES

Aarts, Emile, & Korst, Jan (1989). Simulated annealing and Boltzman machines. New York: John Wiley and Sons.

Aarts, E. H. L., & Van Laarhoven, P. J. M. (1985). Statistical cooling: A general approach to combinatorial optimization problems, Philips Journal of Research, 40(4), 193-226.

Adrabinski, A., & Syslo, M. M. (1983). Computational experiments with some approximation algorithms for the traveling salesman problem, Zastos. Mat., 18, 91-95.

Ashour, S. (1970a). An experimental investigation and comparative evaluation of flow-shop scheduling techniques, Operations Research, 12, 541-549.

Ashour, S. (1970b). A branch and bound algorithm for the flow-shop scheduling problem, A.I.I.E. Transactions, 2, 172-176.

Ball, M., & Magazine, M. (1981). The design and analysis of heuristics, Networks, 11, 215-219.

Banerjee, P., & Jones, M. (1986). A parallel simulated annealing algorithm for standard cell placement on a hypercube computer, Proceedings of the IEEE International Conference on Computer-Aided Design, Santa Clara, Ca, 34-37.

Beaton, R., Adams, M. B., James, V. A., & Harrison, J. S. (1987). Real-time mission and trajectory planning (technical report 1987-8). The Charles Stark Draper Laboratory, Cambridge, Ma.

Bellman, R. E., Esogbue, A. O., & Nabeshima, I. (1982). Mathematical aspects of scheduling and applications. New York: Pergamon Press.

Bellman, R. E. (1957). Dynamic programming. New Jersey: Princeton University Press.

Bellman, R. E. (1962). Dynamic programming treatment of the traveling salesman problem, Journal of the Association of Computing Machinery, 9, 61-63.

Birkhoff, G. (1946). Tres observaciones sobre el algebra lineal, Rev. Univ. Nac. Tucuman Ser. A, 5, 147-151.

Bohr, Niels (1913). Old quantum theory, Philosophical Magazine, 26, 1.

Bonomi, Ernesto, & Lutton, Jean-Luc (1984). The N-city traveling salesman problem: statistical mechanics and the metropolis algorithm, SIAM Review, 26(4), 551-568.

Brooks, Daniel A., & Verdini, William A. (1989). Computational experiments with generalized simulated annealing over continuous variables, Simulated Annealing (SA) and Optimization, Syracuse, New York: American Sciences Press.

Burkard, R. E., & Rendl, F. (1984). A thermodynamically motivated simulation procedure for combinatorial optimization problems, European Journal of Operational Research, 17(2), 169-174.

Catthor, F., de Man, H., & Vandewalle, J. (1988). SAMURAI: A general and efficient simulated annealing schedule with fully adaptive annealing parameters, Integration, 6, 147-178.

Cerny, V. (1985). Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm, Journal of Optimization Theory and Applications, 45(1), 41-52.

Cook, S. A. (1971). The complexity of theorem-proving procedures, Proceedings of the Third ACM Symposium on Theory of Computing, Association for Computing Machinery, New York.

Croll, Tim, & Cady, Larry (1988). Dynamic programming algorithm applications to automated mission planning, La Jolla, Ca: Systems Control Technology, Inc.

Crowder, H., & Padberg, M. W. (1980). Solving large-scale symmetric traveling salesman problems to optimality, Management Science, 26, 495-509.

Dantzig, G. B., Fulkerson, D. R., & Johnson, S. M. (1954). Solution of a large-scale traveling salesman problem, Operations Research, 2, 393-410.

Darema-Rogers, F., Kirkpatrick, S., & Norton, V. A. (1987). Parallel algorithms for chip placement by simulated annealing, IBM Journal of Research and Development, 31, 391-402.

Drex1, A. (1988). A simulated annealing approach to the multiconstraint zero-one knapsack problem, Computing, 40, 1-8.

Dreyfus, Stuart E., & Law, Averill M. (1977). The art and theory of dynamic programming. New York: Academic Press, Inc.

Eisen, Martin (1969). Introduction to mathematical probability theory. Englewood Cliffs, New Jersey: Prentice-Hall.

Farquhar, I. E. (1964). Ergodic theory in statistical mechanics. New York: Interscience Publishers, John Wiley and Sons.

Feller, William (1966). An introduction to probability theory and its applications (Vol II). New York: John Wiley and Sons, Inc.

Flood, M. M. (1956). The traveling salesman problem, Operation Research, 4, 61-75.

Forgionne, Guisseppi A. (1986). Quantitative decision making. Belmont, Ca: Wadsworth Publishing Company.

French, S. (1982). Sequencing and scheduling, An introduction to the mathematics of the job-shop. New York: John Wiley and Sons.

Freville, A., & Plateau, G. (1982). Methodes heuristiques performantes pour les problemes en variables 0-1 a plusieurs contraintes en inegalite, publication ANO-91, Universite des Sciences et Techniques de Lille.

Freville, A., & Plateau, G. (1987). Hard 0-1 knapsack test problems for size reduction methods, Universite Paris-Nord.

Fromhold, Albert Thomas, Jr. (1981). Quantum mechanics for applied physics and engineering. New York: Academic Press.

Garey, Michael R., & Johnson, David S. (1979). Computers and intractability: A guide to the theory of NP-completeness. San Francisco: W. H. Freeman and Company.

Gavish, B., & Pirkul, H. (1985). Efficient algorithms for solving zero-one knapsack problems to optimality, Mathematical Programming, 31, 78-105.

Germain, R., & Sriskandarajan, C. (1985). A heuristic for job shop scheduling, Internatiuonal Federation of Automatic Control, Control Sciences and Technology for Development, Proceedings.

Golden, B. L., Bodin, L. D., Doyle, T., & Stewart, W. Jr. (1980). Approximate traveling salesman algorithms, Operations Research, 28, 694-711.

Greene, J. W., & Supowit, K. J. (1986). Simulated annealing without rejection moves, IEEE Transactions on Computer-Aided Design, 5, 221-228.

Hausner, Melvin (1971). Elementary probability theory. New York: Harper and Row, Publishers.

Hillier, Frederick S., & Lieberman, Gerald J. (1980). Introduction to operations research. San Francisco: Holden-Day.

Ignall, E., & Schrage, L. E. (1965). Application of the branch and bound technique to some flow-shop problems, Operations Research, 14, 400-412.

Iosupici, A., King, C., & Breuer, M. (1983). A module interchange placement machine, Proceedings of the IEEE 20th Design Automation Conference, Port Chester, 495-498.

Jaynes, E. T. (1957). Information theory and statistical mechanics, Physics Review, 106, 620-630.

Karp, R. M. (1972). Reducibility among combinatorial problems, in R. E. Miller and J. W. Thatcher (eds.), Complexity of computer computations. New York: Plenum Press.

Kirkpatrick, S. (1984). Optimization by simulated annealing: quantitative studies, The Journal of Statistical Physics, 34, 975-986.

Kirkpatrick, S., Gelatt, C. D. Jr., & Vecchi, M. P. (1983). Optimization by simulated annealing, Science, 220, 671-680.

Kittel, Charles (1969). Thermal physics. New York: John Wiley and Sons.

Kravitz, S. A., & Rutenbar, R. (1987). Placement by simulated annealing on a multiprocessor, IEEE Transactions on Computer-Aided Design, 6, 534-549.

Krylov, Nikolai S. (1979). Works on the foundations of statistical physics. New Jersey: Princeton University Press.

Lawler, E. L., et al. (1985). The traveling salesman problem: A guided tour of combinatorial optimization. Chichester, Great Britain: Wiley and Sons.

Lin, S., & Kernighan, B. W. (1973a). An effective heuristic algorithm for the traveling salesman problem, Operations Research, 21, 498-516.

Lin, S., & Kernighan, B. W. (1973b). An effective heuristic algorithm for the traveling salesman problem, Murray Hill, New Jersey: Bell Telephone Laboratories, Inc.

Nahar, Surendra, Sahni, Sartaj, & Shragowitz, Eugene (1985). Experiments with simulated annealing, 22nd Design Automation Conference, 748-752.

Nahar, Surendra, Sahni, Sartaj, & Shragowitz, Eugene (1986). Simulated annealing and combinatorial optimization, 23rd Design Automation Conference, 293-299.

Nilsson, Nils J. (1980). Principles of artificial intelligence. Palo Alto, Ca: Tioga Publishing Company.

Papadimitriou, C. H., & Steiglitz, K. (1982). Combinatorial optimization: algorithms and complexity. Englewood Cliffs, New Jersey: Prentice-Hall.

Pearl, Judea (1983). On the discovery and generation of certain heuristics, The AI Magazine, Winter/Spring, 25-33.

Pearl, Judea (1984). Heuristics: intelligent search strategies for computer problem solving. Reading, Ma: Addison-Wesley Publishing Company.

Peters, Tom (1987). Thriving on chaos: handbook for a management revolution. New York: Alfred A. Knopf, Inc.

Phillips, Don T., & Garcia-Diaz, Alberto (1981). Fundamentals of network analysis. Englewood Cliffs, New Jersey: Prentice-Hall.

Randelman, R. E., & Grest, G. S. (1986). N-city traveling salesman problem: optimization by simulated annealings, Journal of Statistical Physics, 45(5/6), 885-890.

Reklaitis, G. V., Ravindran, A., & Ragsdell, K. M. (1983). Engineering optimization - methods and applications. New York: John Wiley and Sons.

Riordan, John (1958). An introduction to combinatorial analysis. New York: Wiley and Sons.

Roberts, Fred S. (1984). Applied combinatorics. Englewood, New Jersey: Prentice-Hall.

Romeo, F., Sangiovanni-Vincentelli, A., & Sechen, C. (1984). Research on simulated annealing at Berkeley, Proceedings of the IEEE International Conference on Computer Design, 652-657.

Romeo, F., Sangiovanni-Vincentelli, A., & Sechen, C. (1985). Probabilistic hill climbing algorithms: properties and applications, Proceedings of the 1985 Conference on VLSI, Chapel Hill, NC, 393-418.

Shannon, C. (1948). A mathematical theory of communication, Bell Systems Technical Journal, 379-623.

Shrodinger, E. (1946). Statistical thermodynamics. London: Cambridge University Press.

Skiscim, Christopher C., & Golden, Bruce L. (1983). Optimization by simulated annealing: A preliminary computational study for the traveling salesman problem, Proceedings 1983 Winter Simulation Conference. Arlington, Va, 523-538.

Spira, P., & Hage, C. (1985). Hardware acceleration of gate array layout, Proceedings of the 22nd Design Automation Conference, Las Vegas, 359-366.

Spitzer, F. L. (1964). Principles of random walk. Princeton, New Jersey: Van Nostrand.

Tucker, Alan (1984). Applied combinatorics (2nd ed.). New York: John Wiley and Sons.

Ullman, J. D. (1984). Computational aspects of VLSI. Rockville, MD: Computer Science Press.

Van Laarhoven, P. J. M., & Aarts, E. H. L. (1987). Simulated annealing: theory and applications. Dordrecht, Netherlands: Kluwer Academic Publishers.

Voight, B. F. (1831). Der handlungsreisende (republished (1981)). Kiel, Germany: Verlag Bernd Schramm.

White, D. J. (1969). Dynamic programming. Edinburgh, England: Oliver and Boyd.

Wyllie, G. A. P. (1970). Elementary statistical mechanics. London: Hutchison and Company LTD.

APPENDIX A

TSP PROGRAM CODE

```

1  '***** TSP PROGRAM USING SIMULATED ANNEALING *****
5  CLS:SCREEN 2
6  LINE (0,35)-(620,180),,B
8  LINE (0,0)-(620,33),,B
10 KEY (10) ON                                'OPT INTERRUPT KEY
12 ON KEY(10) GOSUB 1000
15 GM=100
20 N=30                                         'N=CITIES
22 SD=25                                       'RN SEED
25 F=10                                       'GRAPHICS SCALE FACTOR
30 DIM D(N,N), T(N), S(N), C(N),GM(N)
40 T=1                                         'INITIAL TEMP
45 TN=T                                       'RECORDS INIT TEMP
50 C=5                                         'C=COLOR
60 PAINT (10,10),1,C
65 LINE (0,90)-(0,100)
70 D=0
120 FOR I=1 TO N                              'INITIAL CONFIG
130   C(I)=I
140 NEXT I
270 RANDOMIZE SD
273 '*** FILL DISTANCE MATRIX WITH RAND NUMBERS 0-1 *****
275 FOR I=1 TO N
276   FOR J=1 TO N
282     IF I=J THEN D(I,J)=999:GOTO 288
284     D(I,J)=RND                            'D=DISTANCE MATRIX
288   NEXT J
292 NEXT I
295 '***** CALCULATE INITIAL DISTANCE *****
300 D1=D(C(N),C(1))
320 FOR K=1 TO N-1
330   D2=D2+D(C(K),C(K+1))
340 NEXT K
341 TIMER ON                                  'STOP TIMER
342 ON TIMER(600) GOSUB 900
350 D=D1+D2                                  'D=TOTAL DISTANCE
360 D2=0                                      'RESET DISTANCES
370 D1=0
390 I=1                                       'POSITION MARKER
395 '***** PERMUTE *****
400 J=INT(RND*N)+1                            'RAND INT 1-N
410 'IF J>N THEN 400
430 IF J=I THEN 400
460 IF I<J THEN 470 ELSE 500
465 '***** SET ENDPOINTS *****
470 IMIN=I
480 JMAX=J
490 GOTO 540
500 IMIN=J
510 JMAX=I
530 '***** SET TEMPORARY ROUTE *****
540 FOR K=1 TO I-1

```

```

550   T(K)=C(K)
560 NEXT K
570 FOR K=0 TO JMAX-IMIN
580   T(IMIN+K)=C(JMAX-K)
590 NEXT K
600 FOR K=JMAX+1 TO N
610   T(K)=C(K)
620 NEXT K
630 D1=D(T(N),T(1))
640 FOR K=1 TO N-1
650   D2=D2+D(T(K),T(K+1))
660 NEXT K
670 DP=D1+D2           'CALC TEMPORARY DISTANCE
680 D2 = 0
690 D1=0
710 IF DP < D THEN 760           'ACCEPT
715 '***** APPLY SIM ANNEALG ACCETANCE TEST *****
720 RN=RND
730 X=EXP((D-DP)/T)
750 IF RN<X THEN 760 ELSE 872
760 FOR K=1 TO N           'UPDATE CURRENT CONFIG
770 C(K)=T(K)
790 NEXT K
800 D=DP           'UPDATE DISTANCE
801 '***** GRAPHICS ROUTINE *****
803 NX=NX+2
804 ND=D*F
806 LINE -(NX,ND)
808 IF D<GM THEN GOSUB 2000           'CHECK FOR GLOBAL MIN
840 A=A+1           'INCREMENT NO. OF REPS
850 IF A=15 THEN 852 ELSE 871           'CHECK NO. REPS
852 T=T*.9           'DECREMENT TEMP
854 LINE -(NX,180)           'GRAPHICS-DISPLAYS T UPDT
856 LINE -(NX,ND)           'GRAPHICS-CURRENT CONFIG
860 '***** COLOR ROUTINE *****
861 IF T<.8*TN THEN C=1
862 IF T<.2*TN THEN C=3
863 IF T<.3*TN THEN C=2
864 IF T<.4*TN THEN C=14
865 IF T<.5*TN THEN C=12
866 IF T<.6*TN THEN C=4
867 IF T<.8*TN THEN C=13
869 PALETTE 1,C
870 A=0           'RESET NO. REPS COUNTER
871 R=0           'RESET NO. TRIALS COUNTER
872 R=R+1
874 IF R=100000! THEN 900
880 I=I+1           'INCREMENT MARKER
890 IF I<N THEN 400 ELSE 390
895 '***** PRINT RESULTS *****
900 PRINT "DISTANCE = " GM
905 PRINT "CURRENT DISTANCE = " D

```

```
908 PRINT "TEMP = "T
910 'PRINT "ROUTE : "
920 FOR I=1 TO N
930 '  PRINT "C("I")="C(I)
940 NEXT I
950 END
1000 '***** F10 INTERRUPT - PRINTS CURRENT OPTIMAL *****
1010 PRINT "CURRENT OPTIMAL = " GM
1020 RETURN
2000 '***** UPDATE GLOBAL MIN (GM) *****
2002 FOR K=1 TO N
2004  GM(K)=T(K)
2006 NEXT K
2008 GM=D
2010 RETURN
```

APPENDIX B

ASSIGNMENT PROBLEM CODE

```

10 ***** ASSIGNMENT PROBLEM *****
20 N=50
30 DIM C(N),T(N),M(N,N)
40 T=5
50 ***** DATA *****
60 FOR I=1 TO N
70   FOR J=1 TO N
85     M(I,J)=INT(RND*10)+1
90   NEXT J
100 NEXT I
360 ***** STARTING SOL'TN *****
370 FOR I=1 TO N
375   MIN=100
380   FOR J=1 TO N
385     IF M(I,J)>=MIN THEN 450
390     FOR K=1 TO N
400       IF J=C(K) THEN 450
410     NEXT K
420     IF M(I,J)=1 THEN C(I)=J:GOTO 460
430     MIN=M(I,J)
440     C(I)=J
450   NEXT J
460 NEXT I
470 FOR I=1 TO N
480   T(I)=C(I)
485   PRINT "C("I")= " C(I)
490 NEXT I
491 FOR I=1 TO N
492   D=D+M(I,C(I))
493 NEXT I
495 PRINT "D= "D
500 TIMER ON
510 ON TIMER(180) GOSUB 2010
520 ***** PERMUTE, SWAP J & K *****
530 TD=0
540 J=INT(N*RND)+1
550 K=INT(N*RND)+1
560 IF K=J THEN 550
570 SWAP T(J),T(K)
580 DL=M(J,T(J))+M(K,T(K))-M(J,C(J))-M(K,C(K))
590 IF DL<=0 THEN 1000
600 X=EXP(-DL/T)
610 RN=RND
620 IF X>RN THEN 1000
625 SWAP T(J),T(K)
630 GOTO 530
1000 ***** ACCEPT CHANGE *****
1010 FOR I=1 TO N
1020   C(I)=T(I)
1030 NEXT I
1040 D=D+DL

```

```
1050 T=T*.9
1060 GOTO 530
2000 ***** RESULTS *****
2010 SWAP C(J),C(K)
2040 PRINT "D= "D
2045 BEEP:BEEP:BEEP
2050 END
```


APPENDIX C

JOB-SHOP (ZERO-ONE PROGRAMMING) CODE

```

10 '*** BINARY PROGRAMMING PROBLEM - USING SIM ANNEALG ***
20 N=3
30 M=3
35 T=2.3
40 DIM C(N),X(N),A(M,N),R(M),RHS(M)
45 '***** DATA *****
50 C(1)=2
60 C(2)=1.5
70 C(3)=3.4
80 FOR J=1 TO N
90   X(J)=0
100 NEXT J
110 FOR I=1 TO M
120   FOR J=1 TO N
130     A(I,J)=0
140   NEXT J
150 NEXT I
160 A(1,1)=2
170 A(1,2)=3
180 A(2,2)=2
190 A(2,3)=3
200 A(3,1)=1
210 A(3,3)=2
220 RHS(1)=6
230 RHS(2)=4
240 RHS(3)=5
250 ' ***** GENERATION OF INITIAL SOLUTION *****
260 FOR J=1 TO N
270   X(J)=1
280   GOSUB 1000
290 NEXT J
300 FOR I=1 TO N
310   CP=CP+C(I)*X(I)
320   PRINT "X("I")= "X(I)
330 NEXT I
335 PRINT "CP= "CP
340 TP=0
350 ' ***** PERMUTE *****
360 J=INT((RND(1)*N)+1)
370 IF X(J)=1 THEN 360
380 X(J)=1
390 GOSUB 1000
400 IF X(J)=0 THEN 500
405 CP=CP+C(J)
410 GOTO 6000
500 ' ***** CHECK FOR SWAP J (IN) FOR K (OUT) *****
510 K=INT((RND(1)*N)+1)
520 IF X(K)=0 THEN 510
530 X(K)=0
540 X(J)=1
550 GOSUB 1000

```

CHECK FOR FEAS

```

560 IF X(J)=0 THEN 5000
570 D=C(J)-C(K)
580 IF D>=0 THEN 700
590 X=EXP(D/T)
600 RN=RND(1)
610 IF RN>X THEN 5000
700 X(K)=0
710 X(J)=1
720 CP=CP+D
730 GOTO 6000
990 REM ***** FEAS CHECK *****
1000 FOR H=1 TO M
1010   R(H)=0
1020 NEXT H
1030 FOR H=1 TO M
1040   FOR L=1 TO N
1050     R(H)=R(H)+A(H,L)*X(L)
1060   NEXT L
1070   IF R(H)<=RHS(H) THEN 1080 ELSE 1100
1080 NEXT H
1090 GOTO 1110
1100 X(J)=0
1110 RETURN
1140 '***** TP UPDATE *****
1150 TP=0
1155 FOR J=1 TO N
1160   TP=TP+C(J)*X(J)
1170   PRINT "X("J") = "X(J)
1180 NEXT J
1190 PRINT "TP = "TP
1200 RETURN
5000 '***** CHECK TO DROP VAR K *****
5010 PK=EXP(-C(K)/T)
5020 RN=RND(1)
5030 IF RN>=PK THEN 350
5040 X(K)=0
5050 CP=CP-C(K)
5060 GOTO 350
5990 '***** INCREMENT PARAMETERS *****
6000 T=T*.3
6005 IF T<=9.999999E-35 THEN END
6008 GOSUB 1150
6010 GOTO 350

```

APPENDIX D

MINIMUM CUT PROGRAM CODE

```

10  '*** BINARY PROGRAMMING PROBLEM - USING SIM ANNEALG ***
15  '*** TO SOLVE MAX FLOW FORMULATION *****
20  N=10
35  T=6
40  DIM C(N),X(N),A(N,N)
80  FOR J=1 TO N-1
90    X(J)=0
100 NEXT J
105 X(N)=1
110 FOR I=1 TO N
120   FOR J=1 TO N
130     A(I,J)=0
140   NEXT J
150 NEXT I
155  '***** DATA *****
160 A(1,2)=6
170 A(1,4)=8
180 A(2,5)=5
190 A(2,3)=7
200 A(3,5)=4
210 A(4,5)=7
211 A(3,6)=4
212 A(5,6)=3
213 A(5,8)=9
214 A(6,7)=6
215 A(6,9)=4
216 A(8,9)=7
217 A(7,10)=8
218 A(9,10)=5
250  '***** GENERATION OF INITIAL SOLUTION *****
300 FOR I=1 TO N
305   FOR J=I+1 TO N
310     MF=MF+(A(I,J)*(((1-X(I))*X(J))+(X(I)*(1-X(J)))))
325   NEXT J
330 NEXT I
335 PRINT "MF= "MF
337 'TIMER ON 'OPTIONAL TIMER
340 'ON TIMER(60) GOSUB 7000
350  '***** PERMUTE *****
360 J=INT(RND*(N-2))+2 'RN FROM 2 TO N-1
555 IF X(J)=0 THEN X(J)=1 ELSE X(J)=0
559 TF=0
560 FOR I=1 TO N
565   FOR L=I+1 TO N
570     TF=TF+(A(I,L)*(((1-X(I))*X(L))+(X(I)*(1-X(L)))))
575   NEXT L
580 NEXT I
585 IF TF<=MF THEN 1150
590 X=EXP((MF-TF)/T)
600 RN=RND
610 IF X>RN THEN 1150

```

```
620 IF X(J)=0 THEN X(J)=1 ELSE X(J)=0
622 T=T*.9
625 IF T<=.002 THEN 7000
730 GOTO 360
1140 REM ***** MAX FLOW UPDATE *****
1150 MF=TF
1190 PRINT "MF = "MF
5990 '***** INCREMENT PARAMETERS *****
6000 T=T*.9
6005 IF T<=.002 THEN 7000
6010 GOTO 360
6998 '***** PRINT RESULTS *****
7000 FOR I=1 TO N
7010   PRINT "X("I")= "X(I)
7020 NEXT I
7030 PRINT "MF= "MF
7040 END
```

APPENDIX E

FLOW-SHOP CODE

```

10 ***** FLOW-SHOP FOR > 2 MACHINES *****
12 SEED=20
15 T=3                ' INITIAL TEMP
20 N=10               ' N = # JOBS
21 K=2                ' K = # MACHINES
25 NA=1               ' NA=# OF ACCEPTANCES
27 ALPHA=.98          ' TEMP DECREMENT PARAMETER
28 MCI=0               ' MCI=MARKOV CHAIN INCREMENT
32 GMIN=999
35 RANDOMIZE SEED
37 TIME$="0"
38 KEY (10) ON        ' F10 = HOT INTERRUPT KEY
39 ON KEY(10) GOSUB 4000
40 DIM S(N),C(N,K),D(N,K),GS(N)
45 ***** FILL PROCESSING TIME MATRIX *****
50 FOR J=1 TO N
60   FOR M=1 TO K
70     D(J,M)=INT(RND*10)+1
80   NEXT M
90 NEXT J
100 ***** INITIAL CONFIGURATION *****
110 FOR I=1 TO N
112   S(I)=N+1-I
114 NEXT I
170 ***** CMAX *****
180 FOR J=1 TO N
190   C(S(J),1)=C(S(J-1),1)+D(S(J),1)
200 NEXT J
210 FOR M=2 TO K
220   FOR I=1 TO N
230     IF C(S(I),(M-1)) < C(S(I-1),M) THEN GOTO 260
240     C(S(I),M)=C(S(I),(M-1)) + D(S(I),M)
250     GOTO 270
260     C(S(I),M)=C(S(I-1),M) + D(S(I),M)
270   NEXT I
280 NEXT M
283 CLS
285 CMAX = C(S(N),K)
290 PRINT "CMAX = "C(S(N),K)
300 ***** PERMUTE *****
310 A=INT(RND*N)+1
320 B=INT(RND*N)+1
330 IF A=B THEN 320
340 SWAP S(A),S(B)
350 GOSUB 1000
360 IF TMAX < CMAX THEN 2000
370 X=EXP((CMAX-TMAX)/T)
380 IF X>RND THEN 2000
390 SWAP S(A),S(B)
400 GOTO 310
1000 ***** TMAX *****

```



```

1010 FOR J=1 TO N
1020   C(S(J),1)=C(S(J-1),1)+D(S(J),1)
1030 NEXT J
1040 FOR M=2 TO K
1050   FOR I=1 TO N
1060     IF C(S(I),(M-1)) < C(S(I-1),M) THEN GOTO 1090
1070     C(S(I),M)=C(S(I),(M-1)) + D(S(I),M)
1080     GOTO 1100
1090     C(S(I),M)=C(S(I-1),M) + D(S(I),M)
1100   NEXT I
1110 NEXT M
1120 TMAX = C(S(N),K)
1130 RETURN
2000 '***** ACCEPT *****
2005 AC=AC+1
2010 IF AC>=NA THEN GOSUB 3000
2020 CMAX=TMAX
2030 GOSUB 3500
2040 IF T>.002 THEN 310           'STOP CRITERION
2050 END
2999 '***** UPDATE T & NA *****
3000 T=T*ALPHA
3010 AC=0
3020 NA=NA+MCI
3030 RETURN
3500 '***** UPDATE GLOBAL MIN *****
3510 IF CMAX<GMIN THEN 3520 ELSE 3600
3520 GMIN=CMAX
3530 'FOR I=1 TO N           'OPTIONAL
3540 '  GS(I)=S(I)
3550 'NEXT I
3600 RETURN
3999 '***** INTERRUPT FOR GLOBAL MIN *****
4000 LOCATE 4,5
4010 PRINT "CURRENT OPTIMAL = " GMIN,"TIME = "TIME$
4020 RETURN

```

APPENDIX F

CONTINUOUS FUNCTION CODE

```

5  ** SOLVING A COTINUOUS FUNCTION - SIMULATED ANNEALING **
10 K=50                                ,K=RANGE FOR RANDOM #
11 T=10
12 X=0
14 Y=0
16 C=4*X^2 -64*X +3*Y^2 -42*Y + 24      'FUNCTION
17 PRINT "C= "C
20 TIMER ON
21 ON TIMER (120) GOSUB 300              '2 MIN TIMER
22 P=P+1
23 IF P=10 THEN GOSUB 200
28 '***** PERMUTE *****
29 ' *** GEN RANDOM VARIABLE ASSIGNMENTS WITHIN RANGE K ***
30 TX=INT((1+(2*K))*RND)+(X-K)           'TRIAL X VALUE
40 TY=INT((1+(2*K))*RND)+(Y-K)           'TRIAL Y VALUE
50 TC=4*TX^2 -64*TX +3*TY^2 -42*TY + 24
60 IF TC<C THEN 100
70 XPD=EXP((C-TC)/T)
80 RN=RND
90 IF XPD>RN THEN 100 ELSE 22
100 R=R+1
110 T=T*.5
120 X=TX
125 Y=TY
130 C=TC
138 IF R=10 THEN GOSUB 200
140 GOTO 30
200 K=INT(K*.5)                         'REDUCE RANGE FOR VAR VALUES
210 R=0
220 IF K<1 THEN 300 ELSE 230
230 RETURN
300 PRINT "C= "C,"X= "X,"Y= "Y
310 END

```

APPENDIX G

JOB-SHOP SCHEDULING CODE

```

1  '***** JOB-SHOP SCEDULING W/SIM ANNEALG *****
2  CLS
3  FOR I = 1 TO 10: KEY I, "": NEXT I
4  INPUT "THE NUMBER OF MACHINES = ", M
5  INPUT "THE NUMBER OF JOBS = ", J
6  INPUT "THE SEED = ", SEED
7  RANDOMIZE SEED
8  N = M * J
9  INPUT "DO YOU WANT A CONSULTATION SESSION? ", A$
10 IF A$ = "Y" OR A$ = "YES" OR A$ = "y" THEN GOSUB 9000
   ELSE GOSUB 4000
11 DIM S(M, J), P(M, J), TJ(J), TM(M)
12 KEY(9) ON
13 ON KEY(9) GOSUB 6000
14 LOCATE 23, 1
15 PRINT , "F1...STOP", "F2...RESET", "F9...SF DATA",
   "F10...STATUS"
20 KEY(1) ON
21 ON KEY(1) GOSUB 3000
22 KEY(10) ON
23 ON KEY(10) GOSUB 900
24 KEY(2) ON
25 ON KEY(2) GOSUB 4000
26 A = 0
35 '***** PROCESSING DATA *****
40 FOR L = 1 TO M
50   FOR P = 1 TO J
60     P(L, P) = INT(RND * 10) + 1
70   NEXT P
80 NEXT L
85 '***** INITIAL CONFIG. *****
87 K = 1
90 FOR L = 1 TO M
100  FOR P = 1 TO J
110    S(L, P) = K
115    K = K + 1
120  NEXT P
130 NEXT L
140 'DATA 60,25,10,1,30,75,15,1,2,3,5,1,5,10,30,90
145 'DATA 4,10,11,13,7,2,6,15,9,8,1,16,5,12,14,3
149 '***** CALC INITIAL PROCESSING TIME *****
150 FOR K = 1 TO N
160   FOR L = 1 TO M
170     FOR P = 1 TO J
180       IF P > J THEN 600
190       IF S(L, P) = K THEN 200 ELSE 500
200       TJ(P) = TJ(P) + P(L, P)
210       TM(L) = TM(L) + P(L, P)
220       IF TJ(P) > TM(L) THEN 400
230       TJ(P) = TM(L)
240       IF PT < TM(L) THEN PT = TM(L)

```

```

250      GOTO 700
260      TM(L) = TJ(P)
400      TM(L) = TJ(P)
410      IF PT < TJ(P) THEN PT = TJ(P)
420      GOTO 700
500      NEXT P
600      NEXT L
610      PRINT "PT = "PT,"K = "K,"MA= "L,"JOB = "P
700      NEXT K
705      GMIN = PT
707      LOCATE 9, 1
710      PRINT "INITIAL PROCESSING TIME = "; PT; "          "
720      TIME$ = "0"
799      ***** PERMUTE *****
800      IF A > NA THEN 801 ELSE 810
801      T = T * ALPHA
802      IF T < .5 THEN GOTO 3000
803      IF PT < 145 THEN GOTO 900
805      NA = INT(NA * BETA)
806      A = 0
810      M1 = INT(RND * M) + 1
820      J1 = INT(RND * J) + 1
830      M2 = INT(RND * M) + 1
840      J2 = INT(RND * J) + 1
850      IF M1 = M2 AND J1 = J2 THEN 840
857      SWAP S(M1, J1), S(M2, J2)
858      GOTO 5000                      FEAS CHECK
859      TPT = 0
860      FOR P = 1 TO J
861          TJ(P) = 0
862      NEXT P
865      FOR L = 1 TO M
866          TM(L) = 0
867      NEXT L
869      GOSUB 1000
870      IF TPT <= PT THEN GOTO 2000
875      X = EXP((PT - TPT) / T)
880      IF X > RND THEN GOTO 2000
885      SWAP S(M1, J1), S(M2, J2)
890      GOTO 810
899      ***** INTERRUPT - PRINT STATUS *****
900      LOCATE 10, 1
905      PRINT "GLBL MIN = "; GMIN, "TEMP = "; T, "NA = "; NA
910      PRINT "CRNT MIN = "; PT, "TIME = "; TIME$
920      FOR L=1 TO M
930          FOR P=1 TO J
935              PRINT "S(";LP;") = "S(L,P)
940          NEXT P
945      NEXT L
950      RETURN
1000 ***** CALC TRIAL PROCESSING TIME *****
1150 FOR K = 1 TO N

```

```

1160   FOR L = 1 TO M
1170     FOR P = 1 TO J
1180       IF P > J THEN 1600
1190       IF S(L, P) = K THEN 1200 ELSE 1500
1200       TJ(P) = TJ(P) + P(L, P)
1210       TM(L) = TM(L) + P(L, P)
1220       IF TJ(P) > TM(L) THEN 1400
1230       TJ(P) = TM(L)
1240       IF TPT < TM(L) THEN TPT = TM(L)
1250       GOTO 1700
1400       TM(L) = TJ(P)
1410       IF TPT < TJ(P) THEN TPT = TJ(P)
1420       GOTO 1700
1500     NEXT P
1600   NEXT L
1700 NEXT K
1710 RETURN
1999 '***** ACCEPT CURRENT MIN & CHECK GMIN *****
2000 PT = TPT
2010 IF PT < GMIN THEN GMIN = PT
2015 A = A + 1
2020 GOTO 800
2999 '***** PRINT RESULTS & STOP *****
3000 LOCATE 14, 1
3005 PRINT "GLBL MIN = "; GMIN, "TEMP = "; T, "NA = "; NA
3010 PRINT "CRNT MIN = "; PT, "TIME = "; TIME$
3020 END
3999 '***** PARAMETER INPUT *****
4000 LOCATE 18, 1
4010 INPUT "ALPHA = ", ALPHA
4016 INPUT "THE NUMBER OF ACCEPTANCES = ", NA
4017 INPUT "BETA = ", BETA
4018 INPUT "TEMP = ", T
4020 RETURN
4999 '***** FEAS CHECK *****
5000 IF S(1, 1) < S(2, 1) AND S(2, 1) < S(3, 1) AND S(3, 1)
< S(4, 2) THEN 5010 ELSE 885
5010 IF S(2, 2) < S(3, 2) AND S(3, 2) < S(1, 2) AND S(1, 2)
< S(4, 2) THEN 5020 ELSE 885
5020 IF S(3, 3) < S(2, 3) AND S(2, 3) < S(1, 3) AND S(1, 3)
< S(4, 3) THEN 5030 ELSE 885
5030 IF S(4, 4) < S(1, 4) AND S(1, 4) < S(2, 4) AND S(2, 4)
< S(3, 4) THEN 5040 ELSE 885
5040 GOTO 859
5999 '***** SHOP-FLOOR DATA INTERRUPT *****
6000 M3 = INT(RND * M) + 1
6010 J3 = INT(RND * J) + 1
6020 RP = INT(RND * 10) + 1
6030 GMIN = PT
6040 P(M3, J3) = RP
6050 RETURN
8999 '***** CONSULTANT SESSION MODULE *****

```

```
9000 PRINT "YOUR PROBLEM CURRENTLY HAS "; M;"MACHINES AND "
9010 PRINT J; " JOBS. IF THIS IS INCORRECT PLEASE RESTART."
9020 INPUT "ENTER THE AMOUNT OF PROCESSING TIME AVAILABLE
(MIN) "; DT
9030 PRINT "ENTER THE FREQUENCY OF CHANGE IN THE SHOP-FLOOR
DATA"
9040 INPUT "IN THE FORM: CHANGES PER HOUR "; SFD
10000 IF N <= 16 AND DT >= 10 THEN 11000
10010 IF N <= 16 AND DT <= 2 THEN 12000
10020 IF N <= 16 AND DT >= 2 AND DT < 10 AND SFD > 15 THEN
13000
10030 IF N <= 16 AND DT > 2 AND DT < 10 AND SFD <= 15 THEN
14000
10040 IF N >= 64 AND DT >= 10 AND SFD > 15 THEN 15000
10050 IF N >= 64 AND DT >= 10 AND SFD <= 15 THEN 16000
10070 IF N >= 64 AND DT < 10 THEN 17000
10080 IF N > 16 AND N < 64 AND DT >= 10 AND SFD > 15 THEN
15000
10090 IF N > 16 AND N < 64 AND DT >= 10 AND SFD <= 15 THEN
18000
10100 IF N > 16 AND N < 64 AND DT <= 2 THEN 17000
10110 IF N > 16 AND N < 64 AND DT > 2 AND DT < 10 THEN 12000
10999 '***** SET PARAMETERS *****
11000 T = N * 2.5: ALPHA = .9: BETA = 1.1: NA = 10: GOTO
19000
12000 T = N * .5: ALPHA = .9: BETA = 1: NA = 1: GOTO 19000
13000 T = N: ALPHA = .95: BETA = 1: NA = 1: GOTO 19000
14000 T = N * 2: ALPHA = .9: BETA = 1.1: NA = 10: GOTO 19000
15000 T = N * .75: ALPHA = .9: BETA = 1: NA = 1: GOTO 19000
16000 T = N * .75: ALPHA = .9: BETA = 1.05: NA = 21: GOTO
19000
17000 T = N * .5: ALPHA = .8: BETA = 1: NA = 1: GOTO 19000
18000 T = N * 2: ALPHA = .9: BETA = 1.05: NA = 21: GOTO
19000
19000 RETURN
```


APPENDIX H

ADDITIONAL FORMULATIONS

APPENDIX H

Appendix H presents formulations for zero-one programming, continuous functions, and linear programming problems. Results from the literature for various problems follows the formulation section.

A. Formulations

1. Zero-One Programming

Drex1 (1988) formulates a 0-1 multiconstraint knapsack problem as:

$$\text{maximize } z = \sum_{j=1}^n c_j x_j$$

$$\text{subject to } \sum_{j=1}^n a_{ij} x_j \leq b_i, \quad (i=1, \dots, m), (j=1, \dots, n) \\ x_j \in \{0, 1\}.$$

The variables are partitioned into two sets, those that are equal to zero and those that are equal to one. We begin applying simulated annealing with an initial configuration that has all variables equal to zero. Drex1 recommends setting the initial temperature (control parameter) equal to half the range of the cost coefficients. More formally, $T_0 = .5 * (\max(c_j) - \min(c_j))$, for all j . Permutations from the current configuration are generated by randomly selecting a variable from the set that are equal to zero. This variable is then set equal to one and the constraints

are checked. If the new solution is still feasible the new configuration is accepted. If the new configuration is infeasible, then another random selection is made, but from the set equal to one. A swap of the variables' assignments is performed and the constraints are again checked. Once feasibility is passed, the simulated annealing routine takes place. The algorithm now follows the same general format as the TSP. Better solutions are always accepted. Worse ones are accepted probabilistically, based on the amount of degradation in the objective function and the current value of the control parameter.

A job-shop scheduling problem with n jobs and one machine is polynomially reducible to this form. The equivalent formulation is:

$$\text{maximize } L_{m..x} = \sum_{j=1}^n p_j x_j$$

$$\text{subject to } \sum_{j=1}^n d_{ij} x_j \leq b_i, \quad (i=1, \dots, m), (j=1, \dots, n) \\ x_j \in \{0,1\}.$$

Given a set of processing times $P=\{p_1, p_2, \dots, p_n\}$, due dates d_i , $i=1, 2, \dots, n$, and a number y , is there a schedule such that $L_{m..x} = y$? Here, the objective is to maximize the use of the machine or minimize its idle time for a given processing period.

2. Continuous Functions

In the continuous case, the generation of a neighboring configuration requires the selection of a direction and a distance. Brooks and Verdini (1988) report that this is difficult since the optimal direction and step size are not known prior to any attempt to locate the function's optimum. These two parameters are also very function dependent. Their rule-of-thumb is to select values that result in accepting approximately 60 percent of the detrimental moves. They ran a simulated annealing algorithm against seven test problems and compared results to other stochastic search routines. Results show that the simulated annealing algorithm is competitive with the other routines for the size of problems attempted. The added complexity, however, involved in estimating optimal direction and step size parameters make this approach difficult to use in practical settings. They conclude that the algorithm's sensitivity to these parameter values necessitates substantial user expertise to fine-tune the algorithm.

Our approach to permuting a neighboring configuration is to generate a random number close to the current best value for that variable. By successively narrowing the range of possible random variables, while also narrowing the acceptable limits for the temperature parameter θ , we can gradually hone-in on the optimal values. θ serves to limit, in stages, the width of acceptable "bouncing around." While

large moves are initially tolerated, once we've selected a set of values for the function, we then approach the minimum with only small departures being tolerated. This approach is aimed at avoiding getting "stuck" in a local minimum.

The pseudo-code is shown below:

```

INIT: set values for variables:  $x_0$ ,  $y_0$ ,  $\theta$ ,  $K$ ,  $\alpha$ ,  $\beta$ 
      calculate the cost  $c=f(x_0, y_0)$ 
GEN: while  $P < 20$ 
      generate random values for  $x$  and  $y$  within  $\pm K$  of their
      current values
      calculate the temporary cost  $c_t=f(x_t, y_t)$ 
      if  $c_t < c$  then update
       $x_p=\exp((c-c_t)/T)$ 
      if  $x_p > \text{rnd}(0,1)$  then update, else  $P=P+1$ :goto gen
UPDATE:  $r=r+1$ 
       $\theta=\theta * \alpha$ 
       $x=x_t$ 
       $y=y_t$ 
       $c=c_t$ 
      if  $R=10$  then gosub narrow

```

```

NARROW: K=K * B
        if K<1 then end
        R reset to 0
        return
END:    print c, x, y
        stop

```

Figure H.1 Pseudo-code for a continuous function using simulated annealing.

It is important to know something about the shape of the function we are attempting to optimize in order to correctly and efficiently set the parameters θ , α , K , P , and R . For a given function $f(x,y)$, we set initial values for the variables x , and y , and their bounds $\pm K$. This establishes the range for the random numbers we will be testing. R sets the number of repetitions within each temperature setting (i.e. before decrementing the temperature, $\theta = \theta * \alpha$). Finally, P sets the number of repetitions within a temperature setting when no better solutions are found and B is the decrement factor for K . To prevent the technique from initially finding a local minimum and failing to move to the global minimum we must set P fairly high.

This method was applied to four verifiable functions and in all cases quickly arrived at the minimum. The functions and there optimal values are shown below:

$$1) F(x,y) = 2x^2 - 16x + 3y^2 + 50$$

$$c=18 \quad x=4 \quad y=0$$

$$2) F(x,y) = 4x^2 + 3y^2 - 64x - 42y + 24$$

$$c=-379 \quad x=8 \quad y=7$$

$$3) F(x,y,z) = 4x^2 + 3y^2 + 2z^2 - 64x - 42y - 8z + 24$$

$$c=-387 \quad x=8 \quad y=7 \quad z=2$$

$$4) F(x,y,z,w) = 4x^2 + 3y^2 + 2z^2 - 64x - 42y - 8z + 24 \\ + w^2 - 100w$$

$$c=-2887 \quad x=8 \quad y=7 \quad z=2 \quad w=50$$

Here, Simulated Annealing is seen to offer a good heuristic approach to getting close to optimal solutions in a short amount of time. In the first three cases above, the

optimal values were found in approximately 30 seconds. In case four it took 62 seconds, but a very close solution was found ($c=-2885$, $x=8$, $y=7$, $z=3$, $w=50$) in 22 seconds. This routine could also be used in conjunction with another optimization technique (Hooke-Jeeves Pattern Search, Powell's Conjugate Direction) in an effort to incorporate the speed of the traditional method while using Simulated Annealing to avoid getting trapped in a local minimum.

This use of Simulated Annealing differs from the application to combinatorial optimization problems, in that there are an infinite ^Mnuber of possible solutions to test, whereas in optimizing combinatorial problems there are a finite number of arrangements or states. Also, the movement to a new state (set of variables) depends more heavily on the prior state. Once the integer portion of a variable is set it can not be changed as you move to values of K less than 1. In the discrete case this movement is not restricted, unless it requires movement through a higher cost state and the temperature parameter θ has reached a relatively low value.

The next formulation presented is for the flow-shop problem.

3. A Linear Programming Formulation

The linear programming (LP) example is presented as merely a demonstration of the flexibility of this algorithm. It is not the recommended solution technique for solving LPs. Problem formulation takes the standard approach of the previous examples with the exception that we now include a constraints module. After generating random values for the decision variable coefficients, the constraints are checked to ensure feasibility. If any of the constraints are violated the trial solution is discarded and a new set of coefficient assignments is generated. After a feasible trial solution is obtained the trial objective function value is calculated and the simulated annealing algorithm is resumed. Another modification is made in the generation of coefficient values. A permutation is simply a set of coefficient values. We generate random assignments within a range K. After a number of accepted transitions, we reduce this range around the current values of each decision variable. If X and Y are decision variables with current coefficient values of 3 and 8 respectively and the range K is set at 2, then the random generator will select values for X between 1 and 5 and values for Y between 6 and 10. This is accomplished with the BASIC statement:

$$TX = \text{INT}((1+(2*K))*\text{RND}) + (X-K)$$

The following problems were solved and verified through a commercial software package (LINDO).

1) Min $Z = 2X + 5Y$

Subject To: $X \leq 4$

$Y \leq 3$

$X + 2Y \leq 8$

$X \geq 2$

$Y \geq 1$

Solution: $Z=9, X=2, Y=1$ Time: 3 seconds

2) Max $Z = 3X + 5Y$

Subject To: $X \leq 4$

$2Y \leq 12$

$3X + 2Y \leq 18$

$X, Y \geq 0$

Solution: $Z=36, X=2, Y=6$ Time: 2 seconds

3) Max $Z = 2X + Y$

Subject To: $Y \leq 10$

$2X + 5Y \leq 60$

$X + Y \leq 18$

$3X + Y \leq 44$

$X, Y \geq 0$

Solution: $z=31, X=13, Y=5$ Time: 7 seconds

Solving the same set of problems in LINDO yields the same solutions in approximately 1.2 seconds on an IBM XT.

Problems 2 and 3 are taken from Hillier and Lieberman (1980).

To demonstrate how essentially the same formulation can be used to solve quadratic problems, the first problem is modified to: $\text{Min } Z = 2X^2 + XY + 5Y$ subject to the same set of constraints. The same solution was found. Another problem from Hiller and Lieberman (1980) is shown below.

$$\text{Max } Z = 5X + Y - (X-Y)^2$$

$$\text{Subject To: } X + Y \leq 2$$

$$X, Y \geq 0$$

$$\text{Solution: } Z=7, X=1.5, Y=.5$$

This again is simply a demonstration of the flexibility of the simulated annealing algorithm.

B. Results From the Literature

Most of the analytical results found in the literature are derived from the traveling salesman problem. Other problems have been studied, such as chip design and binary programming, but not nearly as extensively as the TSP. Therefore, we shall present the findings with primary emphasis on the TSP.

1. The Traveling Salesman Problem

Cerny (1985) uses the TSP as a test case for evaluating the simulated annealing algorithm's ability to find an optimal solution to carefully designed problems. The structure of the algorithm is presented in Chapter IV. The permutation mechanism is termed a 2-opt transformation. Here, two randomly chosen nodes (cities) are selected as endpoints and the tour between them is reversed.

The first case contains 100 points uniformly distributed on a unit circle. The optimal path is clearly the one that forms a circle, and its length is reported as 6.28 units. The starting point is a randomly chosen path that has length 150.9 units. Simulated annealing is applied starting at a high value for the temperature parameter ($T=.1$). Figure 4.1 shows a plot of the algorithm's performance. Each point corresponds to the length of the current path after every 200 Monte Carlo trials. The picture shows the behavior expected of the physical annealing process. Equilibrium is approached after approximately 4000 trials, with a corresponding average length of 20. After approaching equilibrium, the control parameter (T) is decreased. This has the effect of restricting the size of acceptable higher-cost paths. Cerny reports that after 25,000 trials the optimal path is found with a length of exactly 6.28. This corresponds to finding one of $99!$ possible configurations.

Since this represents a rather special geometry, the author performed the same procedure on an embellishment of the unit circle. Starting with the same configuration of points, Cerny randomly alters the points slightly so that the optimal path is the same as above, but the geometry is completely different. This is shown below:

$$D(i,j) = \tilde{D}(i,j) + P_i + P_j, \quad i,j=1,2,\dots,100,$$

where $\tilde{D}(i,j)$ are the distances given by the previous example

and $\{P_i\}_1^{100}$ = set of randomly chosen numbers.

Cerny then sets up a test case to see if the algorithm eliminates quickly configurations that are far from optimal. He distributes 200 points randomly within two unit squares, which are ten units apart. Odd points lie in one square and even points in the other. The starting configuration is arranged 1,2,3,...,199,200. The associated path has 200 jumps across squares. Cerny reports that after 12,000 trials at a temperature setting of $T=0.01$ the algorithm correctly eliminated 198 unnecessary jumps.

The intent of the past three examples is to illustrate that the algorithm does eventually converge to optimality. Obviously, for practical implementation we must set the

parameters to find near optimal solutions in less than 12,000 Monte Carlo trials.

In a real industrial application, Lin and Kernighan (1973) used simulated annealing to route a laser drill to 318 hole positions. Numerically controlling a drilling machine efficiently through a set of hole positions is a TSP problem. If the drilling time outweighs greatly the travel time then there is little advantage associated with an optimal path. However, with drilling being performed by pulsed laser, almost all of the cost is in travel time. The problem was solved earlier by hand (by R. Habermann). The tour found by simulated annealing is quite different than the hand derived solution and is .85 inches shorter (41.883 inches versus 42.739 inches). Although the difference is not great, if the drilling procedure is repeated by several machines per day, the savings can be substantial.

Skiscim and Golden (1983) investigate the performance for different size TSPs and compare it to local search heuristics. A portion of their results are shown in Tables H.1 and H.2.

Table H.1

Summary of Computational Results (Skiscim, 1983)

No. of Nodes	Best Known Sol'tn	Swaps/ Epoch						
			Base (%over)	CPU (secs)	+1	CPU	-1	CPU
60	290.28	50	4.69	290.8	3.39	181.0	.0	829.9
70	300.21	50	.0	1296.9	5.96	1314.2	.7	1217.2
75	303.09	50	5.3	1000.9	3.66	984.6	2.2	921.4
80	314.15	5	1.9	401.1	9.22	509.8	7.4	455.6
85	323.45	25	8.6	1002.1	2.31	1142.7	.2	1278.8

The column titled Swaps/Epoch needs some explanation. The term epoch refers to the interval between equilibrium testing. The authors perform a simple test to determine if equilibrium is being approached prior to decreasing the temperature parameter. An epoch consists of n attempted transformations of the current path, or swaps. If equilibrium is not attained, based on their test, another n swaps are attempted. Thus a number of epochs elapse prior to reaching equilibrium. Node locations are obtained from a uniform random number generator, filling a rectangular space. Their cooling schedule allows for, a rule of thumb, 25 temperature levels. The +1 case adds one to every temperature greater than one and the -1 case subtracts one

from every temperature greater than one. Thus +1 accepts a greater number of "worse" paths. Table 4.1 only shows entries in the table that contain the best solutions. For example, we do not show 50, 25, 15, and 5 swaps/epoch cases for all of the five TSP instances.

Results regarding the impact of problem size are not too surprising. As the problem size increases, the running time also increases.

Table H.2 shows similar results for a 2-opt heuristic that only accepts configurations that are of shorter distances. The algorithm is started from 15 different initial configurations.

Table H.2

Repeated Application of the 2-OPT Algorithm (Skiscim, 1983)

No. of Nodes	Best Known Solution	2-OPT Runs	Best (% over)	CPU (secs)
60	290.28	5	1.53	200.12
		10	1.53	403.31
		15	0.00	604.92
70	300.21	5	1.46	380.14
		10	0.69	766.19
		15	0.00	1152.34
75	303.09	5	3.86	533.25
		10	2.86	1056.50
		15	1.51	1598.25
80	314.15	5	2.41	673.45
		10	0.81	1373.43
		15	0.81	1905.98
35	323.45	5	3.10	737.48
		10	1.56	1495.93
		15	0.82	2257.08

The authors conclude that numerous efforts to find a set of parameters failed to arrive at an annealing schedule that performed consistently well. Repeated application of the 2-opt algorithm appears to outperform simulated annealing for the smaller size TSPs. The 2-opt appears to have a barrier of about .8 percent above optimal that is difficult to break through. The authors recommend further testing with a different density function for the probability of acceptance and investigating probabilistically accepting small decreases in the objective function.

Kirkpatrick (1984) examines the potential for using simulated annealing on very large TSP instances. Here, he considers an actual industrial engineering problem, drilling 6,406 holes in a printed circuit board with an automatically positioned laser. The approach is to begin with a "greedy" heuristic to find the initial path and apply the 2-opt permutation mechanism to rearrange current paths. After 20 minutes of cpu time its path was 25 percent shorter than the one arrived at using the greedy heuristic alone.

2. Zero-One Programming

The zero-one programming problem is described by looking at a prototype knapsack problem. Here, the hiker must choose items to pack in his knapsack. Each item has an associated value to him and an associated weight and volume. The objective is for him to maximize the total value of all items packed subject to weight and volume restrictions. The formulation is shown below:

$$\text{Maximize } z = \sum_{j=1}^n c_j x_j$$

$$\text{Subject to: } \sum_{j=1}^n a_{ij} x_j \leq b_i \quad (i=1, \dots, m), (j=1, \dots, n), \\ x_j \in \{0, 1\}$$

Drex1 (1988) examines the performance of simulated annealing compared to a greedy heuristic that only accepts transitions to more optimal configurations. Table H.3 shows results

from 57 test problems taken from Freville and Plateau (1982, 1987). PROEXC (probabilistic exchange) is the simulated annealing routine using the 2-opt permutation. DETEXC (deterministic exchange) only accepts transitions to configurations that are more optimal. In all cases the cpu time is fixed for the comparison. An IBM 3090/200 computer was used for the analysis.

Table H.3

Results From Literature (Drex1, 1987)

Problem	m	n	PROEXC Gap to Opt (%)	CPU time (millisec)	DETEXC Gap to Opt (%)
Wei Shih					
26	5	90	1.00	65.3	1.52
27	5	90	0.00	68.2	3.03
28	5	90	0.26	64.6	0.73
29	5	90	0.98	33.7	1.36
30	5	90	0.48	35.7	2.53
Petersen					
1	10	6	0.00	0.7	0.00
2	10	10	0.00	5.4	0.64
3	10	15	0.00	4.1	0.25
4	10	20	0.16	1.0	1.47
5	10	28	0.08	10.5	0.24
6	5	39	0.32	31.3	1.06
7	5	50	0.56	36.8	8.86
Hansen, Plateau					
1	4	28	0.37	9.3	2.87
2	4	35	0.42	19.0	4.52
Weingartner					
1	2	28	0.00	17.8	0.67
2	2	28	0.63	22.8	4.76
3	2	28	0.27	17.6	1.05
4	2	28	0.00	12.9	2.95
5	2	28	0.49	25.0	3.05
6	2	28	0.13	15.6	0.00
7	2	105	0.10	127.8	0.38
8	2	105	0.68	71.1	1.96
Senju, Toyoda					
1	30	60	0.00	31.4	1.16
2	30	60	0.25	49.4	0.23

Fleisher						
1	10	20	0.78	3.4	2.01	
Freville, Plateau						
1	4	27	0.79	9.2	3.53	
2	4	34	0.41	15.5	5.48	
3	2	19	0.17	10.1	0.00	
4	2	29	0.41	10.1	4.48	
5	10	20	0.79	3.0	2.01	
6	30	40	0.00	12.2	7.73	
7	30	37	0.10	10.5	2.13	

One of the results is that the algorithm finds good solutions very fast. In all cases the solution found is less than one percent away from the optimal solution. The deterministic exchange algorithm usually performed worse with gaps ranging up to about nine percent. This is a partial table, showing results for 33 problems. Looking at results for all 57 problems, the optimal solution was found in 23 of the cases (40 percent). Currently, a restricted branch and bound approach developed by Gavish and Pirkul (1985), which terminates early, seems to work the best for problems with $m \leq 5$.

The author concludes that simulated annealing is fast and easy to implement. Thus, it is a very attractive alternative to existing methods of solving zero-one programming problems Drex1 (1987).

C. Results from the Modified TSP Formulation

Although the traveling salesman problem gives good results in a reasonable amount of time, the major drawback to its implementation is the lengthy search times needed for convergence to the global minimum. Results for various sizes of the TSP are given in table H.4.

Table H.4

Results for the Traveling Salesman Problem5 Nodes

<u>SA-NE</u>	<u>SA-UN</u>	<u>Branch & Bound</u>
19	19	19
15	15	15
16	16	16
19	19	19
12	12	12
16	16	16

8 Nodes

<u>SA-NE</u>	<u>SA-UN</u>	<u>Branch & Bound</u>
19	19	19
20	20	20
15	17	15
14	14	14
14	14	14
26	26	26

10 Nodes

<u>SA-NE</u>	<u>SA-UN</u>	<u>Branch & Bound</u>
21	26	21
18	20	18
17	23	16
20	25	20
15	18	14
27	32	26

At this point it was noted that the search was not spending a sufficient amount of time near the local minimums. In an effort to force it to spend more time searching this region,

modifications to the programs were made. The columns titled SA-ANE and AE-AUN in Table H.5 are the standard forms, but now the algorithms force the route to be near the average for that control parameter setting prior to decrementing the control parameter. A new parameter is also introduced. Beta increases the number of accepted routes that must be found prior to decrementing the control parameter. As the search progresses, the number of accepted routes increases and the algorithm spends a greater amount of time searching the regions near the local minimum. Table H.5 shows the impact of these modifications.

Table H.5

Results for Larger Traveling Salesman Problems12 Nodes

<u>SA-NE</u>	<u>SA-UN</u>	<u>SA-ANE</u>	<u>SA-AUN</u>	<u>Branch & Bound</u>
30	28	26	26	26
25	24	21	22	ERROR
24	26	19	21	19
21	24	19	19	19
24	23	19	20	19
30	32	28	30	28

15 Nodes

<u>SA-NE</u>	<u>SA-UN</u>	<u>SA-ANE</u>	<u>SA-AUN</u>	<u>Branch & Bound</u>
37	37	31	34	29
38	40	31	30	27
31	36	24	26	19
41	42	32	33	28
29	31	27	27	23
35	40	33	29	28

The error entry for the branch and bound routine resulted from an insufficient memory message sent by the computer. Forcing a configuration near the average for each value of the control parameter and requiring more acceptances at lower levels of the search yields better solutions. Comparing the columns SA-NE and SA-ANE (ANE for average, negative exponential) we see that in all 12 cases we achieved shorter routes. Similarly, looking at SA-UN versus SE-AUN we again obtain better solutions in all 12 cases. Comparing the standard form against the version with a modified uniform acceptance function, it appears that the standard form gives better solutions.

When considering that most of the cases returned solutions in less than one minute on an 80386 machine and that the 15 node problems have $15!$ (1.3 trillion) possible solutions, the results indicate potential application in a real time setting. A 112 node TSP returned an initial solution with a distance of 595 units. After 9 minutes and 16 seconds of run time the solution was reduced to 304 units

- almost a fifty percent decrease in less than ten minutes. After approximately 26 minutes the solution was further reduced to 286 units.

Comparing the different versions of simulated annealing, we can see that the modifications result in improved solutions. In both SA-NE and SA-UN, forcing the algorithm to move to an average (quasi-equilibrium) at each level of the control parameter results in improved final solutions. This can be seen by comparing SA-NE with SA-ANE and SA-UN with SA-AUN. Versions with negative exponential and pseudo uniform acceptance functions perform equally well.

Biographical Sketch

James Scott Shedden w [REDACTED]
[REDACTED]. He graduated from high school in Maumee, Ohio in 1973 and attended the United States Air Force Academy from which he received the degree Bachelor of Science in Organizational Behavior in June 1977. Upon graduation, he received a commission in the USAF. In December of 1984 he was awarded a Master of Science Degree from the Air Force Institute of Technology in Dayton, Ohio. His most recent assignment was program manager in the B-1B Bomber Program Office, Wright-Patterson Air Force Base, Ohio. In August, 1987 he began his doctoral studies at Arizona State University. He is currently the liaison officer for 37 Air Force officers assigned to Arizona State University.